
MPC5121e Microcontroller Reference Manual

Devices Supported:
MPC5121e
MPC5123

Document Number: MPC5121ERM
Rev. 2
09/2008

How to Reach Us:

Home Page:

www.freescale.com

E-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 26668334
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The ARM POWERED logo is a registered trademark of ARM Limited. ARM7TDMI-S is a trademark of ARM Limited. Java and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. The PowerPC name is a trademark of IBM Corp. and is used under license. The described product contains a PowerPC processor core. The PowerPC name is a trademark of IBM Corp. and used under license. The described product is a PowerPC microprocessor. The PowerPC name is a trademark of IBM Corp. and is used under license. The described product is a PowerPC microprocessor core. The PowerPC name is a trademark of IBM Corp. and is used under license. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2008. All rights reserved.

MPC5121ERM

Rev. 2
09/2008

Chapter 1 Overview

1.1	Introduction	1
1.1.1	Features	2

Chapter 2 System Configuration and Memory Map (XLBMEN + Mem Map)

2.1	Introduction	1
2.2	Memory Map and Register Definition	1
2.2.1	Local Memory Map Overview and Example	1
2.2.2	Address Translation and Mapping	2
2.2.3	Window into Configuration Space	3
2.2.4	Local Access Windows	3
2.2.5	Local Access Register Memory Map	4
2.2.6	Precedence of Local Access Windows	20
2.2.7	Configuring Local Access Windows	20
2.2.8	Distinguishing Local Access Windows from Other Mapping Functions	20
2.2.9	Outbound Address Translation and Mapping Windows	20
2.2.10	Inbound Address Translation and Mapping Windows	21
2.2.11	PCI Inbound Windows	21
2.2.12	Accessing Internal Memory from External Masters	21
2.3	System Configuration	21
2.3.1	System Configuration Register Memory Map	21

Chapter 3 Signal Descriptions

3.1	Introduction	1
3.1.1	Signals Overview	1
3.2	Output Signal States During Reset	17

Chapter 4 Reset

4.1	Introduction	1
4.2	HRESET Flow	1
4.2.1	Sources	1
4.2.2	Impacts	2
4.3	SRESET Flow	2
4.3.1	Sources	2
4.3.2	Impacts	2
4.4	(PORESET) Power-On Initialization	3
4.5	Reset of Internal Peripherals	3
4.6	Reset Configuration Word (RST_CONF)	4
4.6.1	BMS Operation	6

4.6.2	RTC at Reset	7
4.6.3	JTAG Reset	7
4.6.4	Boot Vector Selection	7
4.6.5	Boot Memory Interface Selection	7
4.6.6	LPC Initialization Sequence	7
4.6.7	NFC Initialization Sequence	8
4.7	Memory Map	9
4.7.1	Reset Configuration Word Low Register (RCWLR)	10
4.7.2	Reset Configuration Word High Register (RCWHR)	11
4.7.3	Reset Status Register (RSR)	12
4.7.4	Reset Mode Register (RMR)	14
4.7.5	Reset Protection Register (RPR)	15
4.7.6	Reset Control Register (RCR)	16
4.7.7	Reset Control Enable Register (RCER)	17
4.8	IO During Reset Assertion	17

Chapter 5

Clocks and Low-Power Modes

5.1	Introduction	1
5.2	System Clock Generation	1
5.2.1	Peripheral Clock Domains	3
5.2.2	Clock Frequency Measurement (CFM) Clock Selection	4
5.2.3	System Oscillator Disable	5
5.2.4	PSC Clock Generation	5
5.2.5	MSCAN Clock Generation	5
5.2.6	RTC Clock Generation	6
5.2.7	SATA Clock Generation	7
5.2.8	USB Clock Generation	7
5.2.9	System PLL and e300 PLL	8
5.3	Clock Control Module	10
5.3.1	Memory Map/Register Definition	10

Chapter 6

AXE System

6.1	Introduction	1
6.1.1	Features	1
6.2	Memory Map and Register Definition	2
6.2.1	Data Memory Map	2
6.2.2	Instruction Memory Map	3
6.2.3	Register Summary	4
6.2.4	Register Descriptions	8
6.3	Functional Description	36
6.3.1	AXE Reset	37
6.3.2	AXE System	37

6.3.3	Data Access Controller	39
6.3.4	DMA	40
6.3.5	Interrupt Controller	42
6.3.6	FIFOs for Inter-Processor Communication	43
6.3.7	Interrupt Enable/Pending and Clear/Set Registers for FIFO1, FIFO2, and Soft Interrupts	44

Chapter 7

Byte Data Link Controller (BDLC)

7.1	Introduction	1
7.1.1	Features	3
7.2	External Signal Description	3
7.3	Memory Map and Register Definition	3
7.3.1	Memory Map	3
7.3.2	Register Summary	4
7.4	Functional Description	21
7.4.1	J1850 Frame Format	21
7.4.2	J1850 VPW Symbols	23
7.4.3	MUX Interface	35
7.4.4	Protocol Handler	37
7.4.5	Transmitting A Message	39
7.4.6	Receiving A Message	44
7.4.7	Transmitting an In-Frame Response (IFR)	48
7.4.8	Receiving An In-Frame Response (IFR)	57
7.4.9	Special BDLC Module Operations	59
7.5	Initialization Information	61
7.5.1	Initializing the Configuration Bits	61
7.5.2	Exiting Loopback Mode and Enabling the BDLC Module	62
7.5.3	Enabling BDLC Interrupts	63

Chapter 8

Clock Frequency Measurement (CFM)

8.1	Introduction	1
8.1.1	Overview	1
8.1.2	Features	1
8.2	Memory Map and Register Definition	2
8.2.1	Memory Map	2
8.2.2	Register Descriptions	3
8.3	Functional Description	5
8.4	Application Example	5

Chapter 9

CPU e300 Core Power Architecture

9.1	Introduction	1
-----	--------------	---

9.2	e300c4lp Processor Core Functional Overview	1
9.3	e300c4lp Core Reference Manual	2
9.4	Unsupported e300c4lp Core Features	2
9.4.1	Instructions	2
9.4.2	CSB Parity	2
9.4.3	Performance Monitor Event	3

Chapter 10

CSB Arbiter and Bus Monitor

10.1	Introduction	1
10.1.1	Features	1
10.2	Memory Map/Register Definition	2
10.2.1	Register Descriptions	3
10.3	Functional Description	18
10.3.1	Arbitration Policy	18
10.3.2	Bus Error Detection	21
10.4	Initialization/Applications Information	24
10.4.1	Initialization Sequence	24
10.4.2	Error Handling Sequence	24

Chapter 11

Direct Memory Access (DMA)

11.1	Introduction	1
11.1.1	Features	1
11.2	Memory Map and Register Definition	2
11.2.1	Register Descriptions	6
11.3	Initialization/Application Information	37
11.3.1	DMA Initialization	37
11.3.2	DMA Programming Errors	37
11.3.3	DMA Arbitration Mode Considerations	38
11.3.4	DMA Transfer	39
11.3.5	TCD Status	42
11.3.6	Channel Linking	43
11.3.7	Dynamic Programming	44

Chapter 12

Display Interface Unit (DIU)

12.1	Introduction	1
12.1.1	Features	1
12.1.2	Modes of Operation	1
12.2	External Signal Description	2
12.3	Memory Map and Register Definition	3
12.3.1	Memory Map	3
12.3.2	Register Summary	4

12.3.3 Register Descriptions	8
12.4 Functional Description	32
12.4.1 Area Descriptor	32
12.4.2 Area Descriptor Format	34
12.4.3 Pixel Structure	42
12.4.4 Pixel Format Conversion	43
12.4.5 Alpha Blending	44
12.4.6 Chroma Keying	44
12.4.7 Gamma Correction	45
12.4.8 Cursor	46
12.4.9 Write Back Operation	47
12.4.10 Color Bar Generation	48
12.4.11 Interrupt Generation	48
12.4.12 Dynamic Priority Generation	49
12.4.13 Display Signal Timing	50
12.5 Initialization/Application Information	51
12.5.1 DIU Initialization	51
12.5.2 Controlling DIU Planes after the DIU is Enabled	52
12.5.3 Synchronize With the Host	52
12.5.4 Recover From Parameter Error	53
12.5.5 Recover From Underrun Error	53

Chapter 13

DRAM Controller

13.1 Introduction	1
13.1.1 Overview	1
13.2 Features	3
13.3 Memory Map and Register Definition	4
13.3.1 Memory Map	4
13.3.2 Register Descriptions	5
13.4 Functional Description	26
13.4.1 Interfacing with the DRAM	26
13.4.2 Programming DRAM Device Internal Configuration Register	27
13.4.3 DRAM Command Engine	27
13.4.4 Write Buffer	27
13.4.5 Timing Manager	28
13.4.6 DRAM Read Block and DRAM Write Block	28
13.4.7 Bus Interface	28

Chapter 14

DRAM Controller Priority Manager

14.1 Introduction	1
14.1.1 Features	2
14.2 Bus Connections	2

14.3	Memory Map and Register Definition	2
14.3.1	Memory Map	2
14.3.2	Register Descriptions	4
14.4	Functional Description	18
14.4.1	Description of Operation — Overview	18
14.4.2	Block Diagram	19
14.4.3	Congestion Detector	20

Chapter 15

External Memory Bus (EMB)

15.1	Introduction	1
15.1.1	Overview	1
15.1.2	Features	1
15.2	Functional Description	1
15.2.1	EMB Mux	1

Chapter 16

Fast Ethernet Controller (FEC)

16.1	Introduction	1
16.1.1	FEC Top Level	1
16.1.2	Features	3
16.1.3	Modes of Operation	3
16.2	External Signal Description (Off Chip)	4
16.2.1	I/O Signal Overview	4
16.2.2	Detailed Signal Descriptions	5
16.3	Memory Map and Register Definition	9
16.3.1	Overview	9
16.3.2	Top-Level Module Memory Map	10
16.3.3	Detailed Memory Map – Control/Status Registers	10
16.3.4	MIB Block Counters Memory Map	12
16.3.5	Register Descriptions	14
16.4	Initialization Information	45
16.4.1	Initialization (Prior to Asserting ETHER_EN)	45
16.5	Buffer Descriptors	46
16.5.1	Driver/DMA Operation with Buffer Descriptors	46
16.5.2	Ethernet Receive Buffer Descriptor (RxBD)	48
16.5.3	Ethernet Transmit Buffer Descriptor	50
16.6	Network Interface Options	52
16.6.1	FEC Frame Transmission	53
16.6.2	FEC Frame Reception	54
16.6.3	Ethernet Address Recognition	55
16.6.4	Full-Duplex Flow Control	60
16.6.5	Inter-Packet Gap Time	61
16.6.6	Collision Handling	61

16.6.7 Internal and External Loopback	61
16.6.8 Ethernet Error-Handling Procedure	62
16.6.9 Transmission Errors	62
16.6.10 Reception Errors	63

Chapter 17

General Purpose Timers (GPT)

17.1 Introduction	1
17.1.1 Modes of Operation	1
17.1.2 Detailed Signal Descriptions	2
17.2 Memory Map and Register Definition	2
17.2.1 Memory Map	2
17.2.2 Register Descriptions	4
17.3 Functional Description	11
17.3.1 Input Capture Mode	12
17.3.2 Changing Sub-Modes	14
17.3.3 Output Compare	14
17.3.4 Force Output Low Immediately	14
17.3.5 Output Pulse High	14
17.3.6 Output Pulse Low	15
17.3.7 Output Toggle	16
17.3.8 Pulse Width Modulation	16
17.3.9 Simple GPIO	19

Chapter 18

General Purpose I/O (GPIO)

18.1 Introduction	1
18.2 Features	1
18.3 Memory Map/Register Definition	2
18.3.1 Register Descriptions	3
18.4 Functional Description	10

Chapter 19

IIM/Fusebox

19.1 Introduction	1
19.2 Overview	1
19.2.1 Features	1
19.2.2 Modes of Operation	1
19.3 Memory Map and Register Definition	1
19.3.1 Memory Map	2
19.3.2 Register Descriptions	2
19.4 Functional Description	15
19.4.1 Fuse Bank 0	15
19.4.2 Fuse Bank 1	15

Chapter 20

Integrated Programmable Interrupt Controller (IPIC)

20.1	Introduction	1
20.1.1	Overview	4
20.1.2	Features	4
20.2	Memory Map/Register Definition	5
20.2.1	Register Summary	5
20.3	Functional Description	35
20.3.1	Interrupt Types	35
20.3.2	Interrupt Configuration	35
20.3.3	Internal Interrupts Group Relative Priority	37
20.3.4	Mixed Interrupts Group Relative Priority	37
20.3.5	Highest Priority Interrupt	37
20.3.6	Interrupt Source Priorities	37
20.3.7	Masking Interrupt Sources	41
20.3.8	Interrupt Vector Generation and Calculation	42
20.3.9	Machine Check Interrupts	42

Chapter 21

Inter-Integrated Circuit (I²C)

21.1	Introduction	1
21.1.1	Overview	1
21.1.2	Features	2
21.1.3	I ² C Controller	3
21.1.4	START Signal	3
21.1.5	STOP Signal	4
21.1.6	Acknowledge	5
21.1.7	Arbitration	7
21.2	External Signal Description	7
21.3	Memory Map and Register Definition	8
21.3.1	Register Descriptions	9
21.4	Initialization Sequence	29
21.5	Transfer Initiation and Interrupt	29
21.5.1	Post-Transfer Software Response	30
21.5.2	Slave Mode	30
21.5.3	Special Note on AKF	30

Chapter 22

IO Control

22.1	Introduction	1
22.1.1	Overview	1
22.1.2	Features	1
22.2	Memory Map and Register Definition	1
22.2.1	Memory Map	1

22.2.2 Register Descriptions	7
------------------------------------	---

Chapter 23 LocalPlus Bus (LPC)

23.1 Introduction	1
23.1.1 Features	2
23.2 Memory Map and Register Definition	3
23.2.1 Register Descriptions	4
23.3 Functional Description	30
23.3.1 Non-Muxed Mode	30
23.3.2 Muxed Mode	38
23.3.3 SCLPC Interface	45
23.3.4 Programmer's Model	46

Chapter 24 MBX Graphics Controller

24.1 Introduction	1
24.1.1 Overview	1
24.1.2 Features	1
24.2 DMA operation	3
24.3 Clocking Architecture of the MBXLITE Core	3

Chapter 25 MSCAN

25.1 Introduction	1
25.1.1 Features	2
25.2 External Signal Description	2
25.2.1 CAN Receiver Input Pins	3
25.2.2 CAN Transmitter Output Pins	3
25.2.3 CAN System	4
25.3 Memory Map and Register Definition	4
25.3.1 Register Summary	5
25.3.2 Register Descriptions	7
25.3.3 Programmer's Model of Message Storage	30
25.4 Functional Description	41
25.4.1 General	41
25.4.2 Message Storage	41
25.4.3 Identifier Acceptance Filter	44
25.4.4 Protocol Violation Protection	48
25.4.5 Clock System	48
25.4.6 Timer Link	50
25.4.7 Modes of Operation	50
25.4.8 Low Power Options	51
25.4.9 Reset Initialization	55

25.4.10	Interrupts	55
25.4.11	Description of Interrupt Operation	55
25.4.12	Interrupt Acknowledge	56
25.4.13	Recovery from Deep Sleep Mode	56
25.4.14	MSCAN Initialization	57
25.4.15	Bus-Off Recovery	57

Chapter 26

NAND Flash Controller (NFC)

26.1	Introduction	1
26.2	Overview	2
26.3	Features	2
26.4	External Signal Description	3
26.5	Memory Map and Register Definition	3
26.5.1	Internal RAM	4
26.5.2	Spare Area Buffer	5
26.5.3	Register Summary	7
26.5.4	Register Descriptions	8
26.6	Functional Description	22
26.6.1	Modes of Operation	22
26.6.2	Bootting From a NAND Flash Device	23
26.6.3	NAND Flash Control	25
26.6.4	NAND Flash Control	25
26.6.5	Flash Clock Diagrams	28
26.6.6	NFC Boot Load Sequence	29
26.6.7	DMA Request Operation	30
26.6.8	RS ECC	31
26.6.9	Address Control	32
26.6.10	RAM Buffer (SRAM)	32
26.6.11	Read and Write Control	33
26.6.12	Endian	33
26.6.13	I/O Pins Sharing	34
26.7	Initialization Information	34
26.7.1	Normal Operation	34
26.7.2	Symmetric Mode – One Flash Clock Cycle Per Input or Output Data Cycle	48
26.7.3	Memory Configuration Examples	53

Chapter 27

Parallel Advanced Technology Attachment (PATA)

27.1	Introduction	1
27.1.1	Features	2
27.1.2	Modes of Operation	2
27.2	External Signal Description	3
27.2.1	Overview	3

27.2.2 Meeting Timing on the ATA Bus	4
27.3 Memory Map and Register Definition	16
27.3.1 Memory Map	16
27.3.2 Register Descriptions	17
27.4 Functional Description	28
27.4.1 Reset	28
27.4.2 Programming ATA Bus Timing and IORDY_EN	28
27.4.3 Access to ATA Bus in PIO Mode	29
27.4.4 Using DMA Mode to Receive Data from the ATA Bus	29
27.4.5 Using DMA Mode to Transmit Data to the ATA Bus	30

Chapter 28

PCI Controller (PCI)

28.1 Introduction	1
28.1.1 Features	1
28.2 External Signal Description	3
28.2.1 Detailed Signal Descriptions	4
28.3 Memory Map and Register Definition	9
28.3.1 Register Descriptions	11
28.4 PCI Interface Functional Description	47
28.4.1 Bus Commands	47
28.5 I/O Sequencer for PCI Subsystem (PCI)	61
28.6 Introduction	61
28.6.1 Features	61
28.7 PCI_IOS Memory Map and Register Definition	62
28.7.1 Register Descriptions	63
28.8 Functional Description	68
28.8.1 Transaction Forwarding	69
28.8.2 PCI Outbound Address Translation	69
28.8.3 Transaction Ordering	70
28.9 DMA for PCI Subsystem (PCI)	71
28.9.1 Features	71
28.9.2 Modes of Operation	72
28.10 External Signal Description	72
28.10.1 Detailed Signal Descriptions	72
28.11 Memory Map and Register Definition	72
28.11.1 Register Descriptions	74
28.12 Functional Description	89
28.12.1 Message Unit	89
28.12.2 DMA Controller	90

Chapter 29

Power Management Control Module (PMC)

29.1 Introduction	1
-------------------------	---

29.1.1 Features	1
29.2 Memory Map and Register Definition	1
29.2.1 Functional description	5
29.2.2 Core PLL Change Mode	10
29.2.3 PRE_DIV Copy Enable Mode	10
29.2.4 Low-Power Configurations	11

Chapter 30

Programmable Serial Controller (PSC)

30.1 Introduction	1
30.2 Memory Map	2
30.2.1 Register Descriptions	3
30.3 PSC Functions Overview	41
30.4 Features	43
30.5 Modes of Operation	44
30.5.1 PSC in UART Mode	44
30.5.2 PSC in Codec Mode	52
30.5.3 PSC in AC97 Mode	62
30.5.4 Local Loop-Back Mode	66
30.5.5 Remote Loop-Back Mode	67

Chapter 31

PSC Centralized FIFO Controller (FIFOC)

31.1 Introduction	1
31.1.1 Features	2
31.1.2 Modes of Operation	2
31.1.3 Register Descriptions	4
31.2 Functional Description	18

Chapter 32

Real Time Clock (RTC)

32.1 Introduction	1
32.1.1 Features	2
32.2 External Signal Descriptions	3
32.3 Memory Map and Register Definition	4
32.3.1 Memory Map	4
32.3.2 Register Descriptions	5
32.4 Functional Description	30
32.4.1 Behavior at Power On	30
32.4.2 Behavior of Wakeup Sources	30
32.4.3 Behavior During Power Off (Hibernation Mode)	31
32.4.4 RTC Response to Target Time Register/Actual Time Count Register and External Wakeup Sources	33
32.4.5 RTC Response to External Wakeup Sources	36

Chapter 33

SATA Controller (SATA)

33.1	Introduction	1
33.1.1	Features	1
33.1.2	Modes of Operation	1
33.2	External Signal Description	2
33.3	Memory Map and Register Definition	3
33.3.1	Register Descriptions	10
33.4	Functional Description	39
33.4.1	Clock	39
33.4.2	Interrupt	40
33.4.3	ATAPI Support	40
33.4.4	PIO Transfers	40
33.4.5	DMA Transfers	41
33.4.6	Power Management	42
33.4.7	DMA Controller	42
33.4.8	Physical Coding Sublayer (PCS)	43
33.4.9	Serial ATA Physical Layer Macro (SATA PHY)	43
33.5	Initialization Information	44

Chapter 34

Secure Digital Host Controller (SDHC)

34.1	Introduction	1
34.1.1	Features	2
34.2	External Signal Description	2
34.2.1	Detailed Signal Descriptions	3
34.3	Memory Map and Register Definition	3
34.3.1	Memory Map	3
34.3.2	Register Descriptions	4
34.4	Functional Description	26
34.4.1	Data Buffers	26
34.4.2	DMA Interface	30
34.4.3	Memory Controller	31
34.4.4	SDIO Card Interrupt	32
34.4.5	Card Insertion and Removal Detection	34
34.4.6	Power Management	35
34.4.7	System Clock Controller	35
34.5	Initialization Information	36
34.5.1	MMC_SD_CLK Control	37
34.5.2	Command Submit – Response Receive Basic Operation	37
34.5.3	Card Identification Mode	38
34.5.4	Card Access	42
34.5.5	Switch Card Mode	45

Chapter 35

Software Watchdog Timer (WDT)

35.1 Introduction	1
35.1.1 Features	1
35.1.2 Modes of Operation	1
35.2 Memory Map/Register Definition	2
35.2.1 Memory Map	2
35.2.2 Register Descriptions	2
35.3 Functional Description	5
35.3.1 Software Watchdog Timer Unit	5
35.3.2 Modes of Operation	7

Chapter 36

Sony/Philips Digital Interface (SPDIF)

36.1 Introduction	1
36.1.1 Features	2
36.2 External Signal Description	2
36.2.1 Pin Signal Descriptions	2
36.2.2 Detailed Signal Descriptions	2
36.3 Memory Map and Register Definition	3
36.3.1 Register Descriptions	4
36.4 Functional Description	26
36.4.1 SPDIF Receiver	26
36.4.2 SPDIF Transmitter	33

Chapter 37

SRAM Memory (MEM)

37.1 Introduction	1
-------------------	---

Chapter 38

Temperature Sensor

38.1 Introduction	1
38.1.1 Normal Operation Mode	1

Chapter 39

Universal Serial Bus Interface with On-The-Go

39.1 Introduction	1
39.1.1 Overview	1
39.1.2 Features	1
39.1.3 Modes of Operation	2
39.2 Memory Map/Register Definitions	2
39.2.1 Register Descriptions	7
39.3 Functional Description	65

39.3.1 System Interface	65
39.3.2 DMA Engine	65
39.3.3 FIFO RAM Controller	65
39.4 OTG Operations	66
39.4.1 Register Bits	66
39.4.2 Hardware Assist	66
39.5 Host Data Structures	68
39.5.1 Periodic Frame List	68
39.5.2 Asynchronous List Queue Head Pointer	70
39.5.3 Isochronous (High-Speed) Transfer Descriptor (iTd)	70
39.5.4 Split Transaction Isochronous Transfer Descriptor (siTD)	74
39.5.5 Queue Element Transfer Descriptor (qTD)	78
39.5.6 Queue Head	84
39.5.7 Periodic Frame Span Traversal Node (FSTN)	88
39.6 Host Operational Model	89
39.6.1 Host Controller Initialization	90
39.6.2 Suspend/Resume	91
39.6.3 Schedule Traversal Rules	93
39.6.4 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries	95
39.6.5 Periodic Schedule	97
39.6.6 Managing Isochronous Transfers Using iTDs	98
39.6.7 Asynchronous Schedule	102
39.6.8 Operational Model for NAK Counter	110
39.6.9 Managing Control/Bulk/Interrupt Transfers via Queue Heads	112
39.6.10 Ping Control	123
39.6.11 Split Transactions	124
39.6.12 Host Controller Pause	152
39.6.13 Port Test Modes	153
39.6.14 Interrupts	153
39.7 Device Data Structures	158
39.7.1 Endpoint Queue Head	158
39.7.2 Endpoint Transfer Descriptor (dTd)	161
39.8 Device Operational Model	163
39.8.1 Device Controller Initialization	163
39.8.2 Port State and Control	164
39.8.3 Bus Reset	166
39.8.4 Managing Endpoints	167
39.8.5 Device Operational Model For Packet Transfers	170
39.8.6 Managing Queue Heads	177
39.8.7 Managing Transfers with Transfer Descriptors	179
39.8.8 Device Error Matrix	181
39.8.9 Servicing Interrupts	182
39.8.10 Deviations from the EHCI Specifications	183
39.9 USB 2.0 PHY with On-The-Go	185
39.9.1 Introduction	185

Chapter 40

Video-In (VIU)

40.1 Introduction	1
40.1.1 Features	2
40.2 External Signal Description	2
40.3 Memory Map and Register Definition	2
40.3.1 Memory Map	2
40.3.2 Register Summary	2
40.3.3 Register Descriptions	5
40.4 Functional Description	13
40.4.1 ITU656	13
40.4.2 Round and Dither	14
40.4.3 DMA and De-interlacing	15
40.4.4 Error Case	16
40.5 Initialization/Application Information	17
40.5.1 Initialization Information	17
40.5.2 Application Information	17

Chapter 1

Overview

1.1 Introduction

The MPC5121e integrated processor provides an exceptional computing platform for multimedia and infotainment vehicle applications for OEM, aftermarket, and commercial products. The MPC5121e is also excellent for any embedded solution that requires graphics, a graphical user-interface, and network connectivity. The MPC5121e has automotive qualification; therefore, all customers can expect competitive cost, quality, reliability, and availability for years to come. The MPC5121e uses the e300 CPU core based on the Power Architecture™ instruction set.

The MPC5121e has an integrated graphics engine, the PowerVR® MBX Lite IP core licensed from Imagination Technologies, which supports 3D acceleration (not available in MPC5123). With a 128-bit interface, this graphics engine has incredible performance. A separate 32-bit RISC auxiliary acceleration engine (AXE) provides additional processing power. This engine has been optimized for audio applications and acceleration of popular media formats including MP3, AAC, WMA, Ogg Vorbis, and others. The AXE can also support sample rate conversion important to speech recognition.

The MPC5121e integrated processor includes multiple cores and multiple buses, helping to avoid high clock rates to obtain high performance. The excellent balance between operating power consumption and performance allows for lower system cost and higher reliability. The low standby power consumption also makes the product suitable for portable applications.

The flexibility of the MPC5121e provides customers a platform for a variety of product applications. Its rich set of integrated peripherals include PCI, SATA, PATA, Ethernet, USB 2.0, CAN, twelve programmable serial controllers, and numerous others. The integrated display controller (DIU) allows for cost-effective support of thin film transistor (TFT) LCD panel displays with up to 1280 x 720 resolution.

Again, the MPC5121e uses the e300 CPU core, with 32 Kbyte instruction cache and 32 Kbyte data cache, based on the Power Architecture instruction set. Wide support of RTOS, software drivers, middleware, and application solutions from mobileGT alliance members is planned when samples become available. This can greatly reduce development lead times and expense while improving software quality.

The many embedded memory buffers help ensure balanced system performance and system bus throughput. The performance of the MPC5121e is enhanced by having well-balanced system resources for the integrated core, graphics and audio engines.

Figure 1-1 shows a top-level block diagram of the MPC5121e.

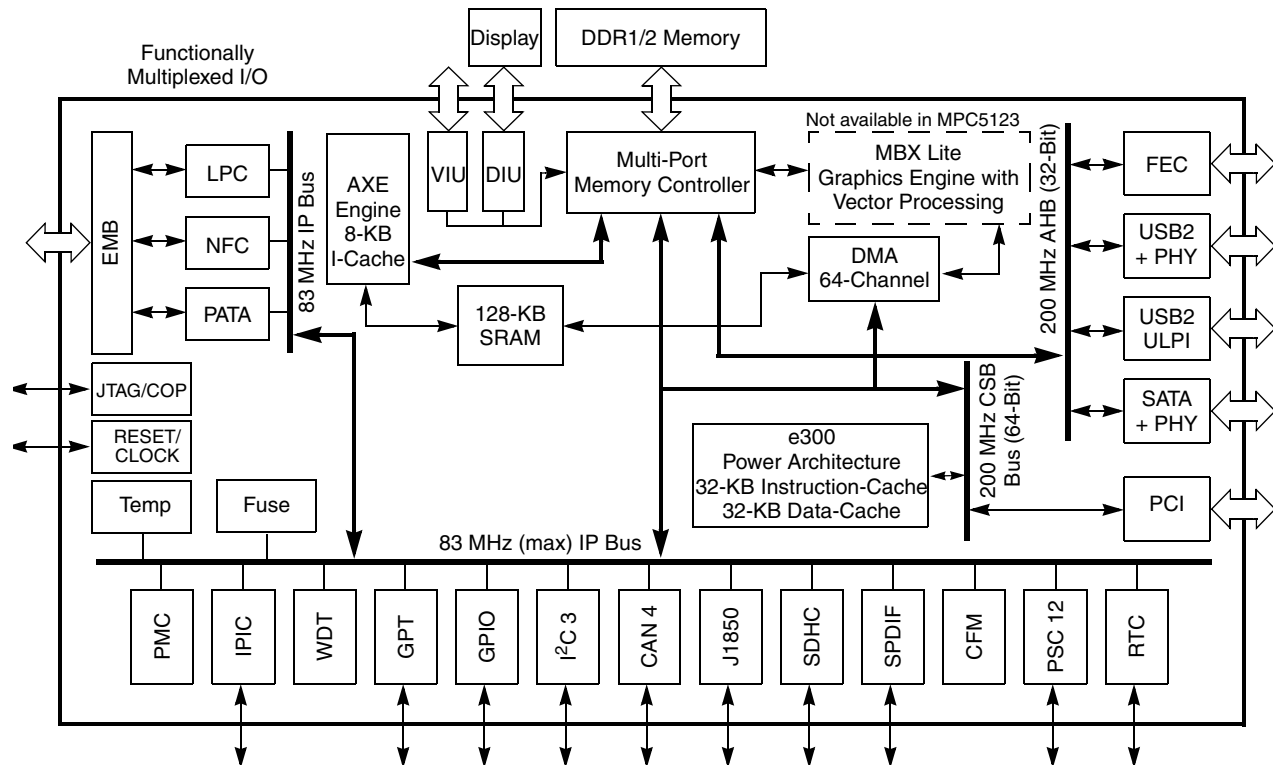


Figure 1-1. MPC5121e Block Diagram

1.1.1 Features

1.1.1.1 Chip-Level Features

Major features of the MPC5121e are as follows:

- e300 Power Architecture processor core
- Power modes include doze, nap, sleep, deep sleep, and hibernate
- AXE – Auxiliary Execution Engine
- MBX Lite – 2D/3D graphics engine (not available in MPC5123)
- DIU – Display interface unit
- DDR1, DDR2, and LPDDR/mobile-DDR SDRAM memory controller
- MEM – 128 Kbyte on-chip SRAM
- USB 2.0 OTG controller with integrated physical layer (PHY)
- DMA subsystem
- EMB – Flexible multi-function external memory bus interface
- NFC – NAND flash controller
- LPC – LocalPlus interface
- 10/100Base Ethernet
- PCI interface, version 2.3

- PATA – Parallel ATA integrated development environment (IDE) controller
- SATA – Serial ATA controller with integrated physical layer (PHY)
- SDHC – MMC/SD/SDIO card host controller
- PSC – Programmable serial controller
- I²C – inter-integrated circuit communication interfaces
- S/PDIF – Serial audio interface
- CAN – Controller area network
- BDLC – J1850 interface
- VIU – Video Input, ITU-656 compliant
- RTC – On-Chip real-time clock
- On-chip temperature sensor
- IIM – IC Identification module

1.1.1.2 Module Features

The following provides more details of modules implemented on the device:

- e300 processor core using the Power Architecture instruction set
 - 32 Kbyte instruction cache
 - 32 Kbyte data cache
 - High-performance, superscalar processor core with a four-stage pipeline
 - Dual-issue processor with integrated floating-point unit and dual integer units
 - Dynamic power management
- MBX Lite graphics block (not available in MPC5123)
 - Dedicated hardware graphics coprocessor
 - Superior 2D and 3D graphics performance
 - Operating system with application programmer interface and drivers
- Display interface unit
 - Supports LCD display resolution up to 1280 × 720
 - Supports refresh rate up to 60 Hz
 - Color depth up to 24 bits per pixel
 - Hardware n-plane accelerated blending
- Video Interface Unit
 - Support from QVGA to XVGA 8-bit/10-bit ITU656 video input
 - YUV to RGB888/565 conversion
 - Internal DMA engine for data transferring to memory
- AXE processor
 - 32-bit RISC coprocessor
 - 8 Kbyte instruction cache, 48-bit fixed point arithmetic, and multiply-accumulate (MAC)

- Supports up to 4x MP3 encode speed
- Software available for many compressed audio formats such as MP3, AAC, Ogg Vorbis, and WMA
- Two USB controllers
 - Two on-chip USB controllers with On-The-Go host/device capability
 - Each supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps)
 - One USB controller with an integrated on-chip physical interface (PHY)
 - Both USB controllers can be accessed through the ULPI interface
- DMA
 - 64-channel on-chip DMA engine with advanced capabilities
 - Supports channel linking and scatter/gather processing
 - Support for external DMA requests
- DDR SDRAM controller
 - Supports 16-bit wide and 32-bit wide DDR1, DDR2, and LPDDR/mobile-DDR SDRAM devices at up to 200 MHz
- 128 Kbyte on-chip SRAM
 - Usable as e300 core scratch pad memory
- PCI interface
 - PCI specification revision 2.3 compliant
 - 32-bit PCI interface support
 - On-chip arbitration supports three external PCI bus masters
 - Support for external DMA requests
- SATA controller with integrated PHY
 - Compliant with SATA 1.0a spec
 - Supports 1.5 Gbps
- Parallel ATA controller
 - Compliant with ATA-6 specification
 - Supports PIO mode 0 to 4
 - Supports MDMA mode 0 to 2
 - Supports UDMA mode 0 to 4
- Ethernet controller
 - Supports 100 Mbps/10 Mbps IEEE 802.3 MII
 - Supports 10 Mbps 7-wire interface
 - IEEE 802.3 full duplex flow control
- NAND flash interface
 - Supports 8-bit-wide and 16-bit-wide NAND flashes
 - Supports booting from NAND flash
 - Supports up to four Kbyte block NAND devices

- Supports four chip selects
- Correction and detection up to eight erroneous symbols
- LocalPlus interface
 - Interface to external memory-mapped or chip-selected devices
 - 32-bit address bus
 - 32-bit data bus
 - Eight chip selects
 - Supports burst mode flash
 - Supports 32-bit ALE-muxed interface
 - Supports up to 42-bit non-muxed interfaces
 - Supports large-packet DMA transfers
- SD/SDIO/MMC card interface
 - Compliant with SD and SDIO specification version 1.x
 - Compliant with MMC card specification
 - 100 Mbps data rate in 4-bit mode
- Four controller area network (CAN) interfaces
 - Implementation of CAN protocol, version 2.0 A/B
 - Programmable wakeup functionality
 - Support of low speed or high speed
- S/PDIF receive and transmit interface
 - S/PDIF receiver operates with incoming frequencies in 32 kHz to 96 kHz range
 - S/PDIF transmitter
- Three inter-integrated circuit communication (I²C) interfaces
 - Three industry-standard I²C interfaces
 - Input digital noise filtering
 - Master and slave modes supported
- Twelve programmable serial controllers
 - Each PSC is a flexible serial communication engine, supporting the following protocols: UART, Codec/PCM, serial audio data, I²S, multi-channel data, SPI, and AC97
 - PSC UART mode
 - PSC Codec mode Master and slave clock support
 - PSC inter-integrated sound interface (I²S mode)
 - PSC SPI mode
 - PSC AC97 mode
- Frequency measurement block
 - Four-Channel frequency measurement for I²S and S/PDIF serial interfaces
- J1850 interface (BDLC)

- SAE J1850 Class B data communications network interface compliant
- ISO-compatible for low speed (< 125 Kbps) serial data communications
- Digital noise filter
- General purpose I/O
 - Up to 32 GPIOs
 - Four GPI available for external wake up
- On-chip real-time clock
 - Real-time clock runs from separate Vbat power domain
 - Tamper bit indicates when Vbat has been removed
 - Programmable alarm
 - Periodic interrupts for one second, one minute, one day
 - Runs with external 32 kHz crystal or external clock source
- IC Identification module (IIM)
 - One fuse bank user space (32 Bytes)
- On-chip temperature sensor
- Hibernation mode
 - CPU and all internal modules are stopped; only Vbat supply is powered
 - Six wakeup sources
 - Hibernation mode power consumption below 20 μ A
- Deep sleep mode
 - All on-chip clocks except the real-time clock are stopped
 - Wakeup from deep sleep mode possible from a number of sources
 - CAN1_RX and CAN2_RX
 - RTC alarm
 - Four external wakeup inputs (GPIO28, GPIO29, GPIO30, and GPI31) on VBAT_RTC domain
 - Power consumption in deep sleep mode below 1 mA
- System timer
 - Real-time clock
 - Eight general-purpose timers
- IEEE 1149.1 compliant JTAG boundary scan

Chapter 2

System Configuration and Memory Map (XLBMEN + Mem Map)

2.1 Introduction

The System Configuration and Memory Map chapter describes the memory map of the MPC5121e and also details information on setting up the system configuration. The memory map is described by 17 Local Access Windows. Some of these memory access windows point to various blocks of the 32-bit memory address space while other memory access windows point to specific peripheral modules. A list of the Local Access Windows is shown in [Table 2-1](#). The registers that set the base address and size of each Local Access Window are presented in [Table 2-2](#).

2.2 Memory Map and Register Definition

2.2.1 Local Memory Map Overview and Example

The MPC5121e provides a flexible local memory map. The local memory map refers to the 32-bit address space seen by the processor as it accesses memory and I/O space. Internal DMA engines also see this same local memory map. All memory accessed by the MPC5121e DDR SDRAM and LocalPlus controllers exists in this memory map, as do all memory-mapped configuration, control, and status registers.

The local memory map of the MPC5121e is defined by a set of 17 local access windows. Each of these windows map a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The LPC windows do not perform any address translation. Each local access window is assigned to a specific target interface as specified in [Table 2-1](#).

Local Access Windows are defined in several different ways. For instance, Local Access Window Number 0 is used for the configuration registers which include the IMMRBAR register and the base address for registers in the various peripheral modules. The block size for Local Access Window 0 is fixed at 1 Mbyte. Local Access Windows 1 – 9 are specified by the Start_Addr and Stop_Addr fields of the respective LocalPlus Access Window Registers for Chip Select Boot and Chip Selects 0 – 7. The base addresses for the three PCI Local Access Windows are specified by the respective PCI Local Access Window Base Address Registers and the size of the windows are specified by the respective PCI Local Access Window Attributes Register. The DDR SDRAM window base address and size is specified by the DDR Local Access Window Base Address Register and the size of the window is specified by the DDR Local Access Window Attributes Register. Window 14 is used for the MBX module. The MBX base address register is located at IMMR + 0xC0. The block size is fixed at 16 Mbytes. Window 15 is used for the SRAM module. The SRAM module base address register is located at IMMRBAR + 0xC4. The block size is fixed at 256 Kbytes. While the SRAM window is 256 Kbytes, the actual size of the SRAM is 128 Kbytes. Window 16

is used for the NAND flash controller. The NAND flash controller base address register is located at IMMR + 0xC8. The block size is fixed at 1 Mbyte.

NOTE

It is generally a programming error to overlap the addressing range of the Local Access Windows. There is nothing to prevent software from programming overlapping addresses.

Changing the value of IMMRBAR changes the location of the Local Access Windows Target Interface. The address of the IMMRBAR Register is the contents of the IMMRBAR register, itself. A Special Purpose Register, SPR311, is specifically provided such that a copy of IMMRBAR can be maintained in a register that does not move in the memory map.

NOTE

It is the responsibility of the system software to properly maintain e300 special purpose register 311 (SPR311 - MBAR) such that its contents are the same as the contents of the IMMRBAR Register.

2.2.2 Address Translation and Mapping

Table 2-1. Local Access Windows Target Interface

Window Number	Offset to IMMR	Target Interface	Comments
0	0x000	Configuration registers (IMMRBAR)	Fixed 1 Mbyte window size
1	0x020	LocalPlus Bus Boot Access Window Reg.	—
2	0x024	LocalPlus Bus CS0 Access Window Reg.	—
3	0x028	LocalPlus Bus CS1 Access Window Reg.	—
4	0x02C	LocalPlus Bus CS2 Access Window Reg.	—
5	0x030	LocalPlus Bus CS3 Access Window Reg.	—
6	0x034	LocalPlus Bus CS4 Access Window Reg.	—
7	0x038	LocalPlus Bus CS5 Access Window Reg.	—
8	0x03C	LocalPlus Bus CS6 Access Window Reg.	—
9	0x040	LocalPlus Bus CS7 Access Window Reg.	—
10	0x060	PCI Local Access Window 0 Base Address Reg.	—
	0x064	PCI Local Access Window 0 Attributes Reg.	—
11	0x068	PCI Local Access Window 1 Base Address Reg.	—
	0x06C	PCI Local Access Window 1 Attributes Reg.	—
12	0x070	PCI Local Access Window 2 Base Address Reg.	—
	0x074	PCI Local Access Window 2 Attributes Reg.	—
13	0x0A0	DDR SDRAM Local Access Window Base Address Reg.	—
	0x0A4	DDR SDRAM Local Access Window Attributes Reg.	—
14	0x0C0	MBX Base Address Reg.	Fixed 16 Mbyte window size

Table 2-1. Local Access Windows Target Interface (continued)

Window Number	Offset to IMMR	Target Interface	Comments
15	0x0C4	SRAM Base Address Register	Fixed 256 Kbyte window size
16	0x0C8	NFC Base Address Register	Fixed 1 Mbyte window size

Three distinct types of translation and mapping operations are performed on transactions in the MPC5121e. These are:

- Mapping a local address to a target interface
- Translating the local 32-bit address to an external address space
- Translating external addresses to the local 32-bit address space

The local access windows perform target mapping for transactions within the local address space. No address translation is performed by the local access windows.

Outbound windows perform the mapping from the local 32-bit address space to the address spaces of PCI, which may be much larger than the local space.

Inbound windows perform the address translation from the external address spaces of PCI to the local address space.

The target mappings created by an inbound window must be consistent with those of the local access windows. That is, if an inbound window maps a transaction to a given local address, a valid local access window must be set independently.

2.2.3 Window into Configuration Space

The internal memory map registers' base address register (IMMRBAR) defines a window that is used to access all memory-mapped configuration, control, and status registers, referred as internal memory map registers or IMMR. The window is always enabled with a fixed size of 1 Mbyte. There is no attributes register associated with Window 0. This window always takes precedence over all local access windows. The IMMRBAR always comes out of reset with a default base address value of 0xFF400000. The value of IMMRBAR can be modified by writing to this register. For more information, see [Section 2.2.5.1.1, "Internal Memory Map Registers Base Address Register \(IMMRBAR\)."](#)

NOTE

Even though Window 0 only uses a 1 Mbyte addressing range, it is recommended that it be treated as a 4 Mbyte addressing range. For example, if IMMRBAR is set to 0xFF40_0000, reserve an address space of 0xFF40_0000-0xFF7F_FFFF. Although it is legal to use the 3 Mbyte address space directly above Window 0, this space may be used in future derivatives of MPC5121e.

2.2.4 Local Access Windows

As demonstrated in the address map overview in [Section 2.2.1, "Local Memory Map Overview and Example,"](#) local access windows associate a range of the local 32-bit address space with a particular target

interface. This allows the internal interconnections of the MPC5121e to route a transaction from its source to the proper target. No address translation is performed. The base address defines the high order address bits that give the location of the window in the local address space. The window attributes enable the window and define its size, while the window number specifies the target interface.

With the exception of configuration space (mapped by IMMRBAR), all addresses used by the system must be mapped by a local access window. This includes addresses that are mapped by PCI inbound windows.

The local access window registers exist as part of the local access block in the system configuration registers. See [Section 2.3.1.1, “System Configuration Registers.”](#) A detailed description of the local access window registers is given in the following sections.

2.2.5 Local Access Register Memory Map

[Table 2-2](#) shows the memory map for the local access registers.

Table 2-2. Local Access Register Memory Map

Local Memory Offset (Hex)	Register	Access	Section/Page
0x000	Internal Memory Map Base Address Register (IMMRBAR) See Table 2-4 , MPC5121e Memory Map for individual module base addresses.	R/W	2.2.5.1.1/2-5
0x004–0x01C	Reserved		
0x020	LocalPlus Boot Access Window register (LPBAW)	R/W	2.2.5.1.3/2-9
0x024	LocalPlus CS0 Access Window register (LPCS0AW)	R/W	2.2.5.1.3/2-9
0x028	LocalPlus CS1 Access Window register (LPCS1AW)	R/W	2.2.5.1.3/2-9
0x02C	LocalPlus CS2 Access Window register (LPCS2AW)	R/W	2.2.5.1.3/2-9
0x030	LocalPlus CS3 Access Window register (LPCS3AW)	R/W	2.2.5.1.3/2-9
0x034	LocalPlus CS4 Access Window register (LPCS4AW)	R/W	2.2.5.1.3/2-9
0x038	LocalPlus CS5 Access Window register (LPCS5AW)	R/W	2.2.5.1.3/2-9
0x03c	LocalPlus CS6 Access Window register (LPCS6AW)	R/W	2.2.5.1.3/2-9
0x040	LocalPlus CS7 Access Window register (LPCS7AW)	R/W	2.2.5.1.3/2-9
0x044–0x05C	Reserved		
0x060	PCI Local Access Window0 Base Address register (PCILAWBAR0)	R/W	2.2.5.1.5/2-11
0x064	PCI Local Access Window0 Attribute register (PCILAWAR0)	R/W	2.2.5.1.6/2-12
0x068	PCI Local Access Window1 Base Address register (PCILAWBAR1)	R/W	2.2.5.1.5/2-11
0x06C	PCI Local Access Window1 Attribute register (PCILAWAR1)	R/W	2.2.5.1.6/2-12
0x070	PCI Local Access Window2 Base Address register (PCILAWBAR2)	R/W	2.2.5.1.5/2-11
0x074	PCI Local Access Window2 Attribute register (PCILAWAR2)	R/W	2.2.5.1.6/2-12
0x078–0x09C	Reserved		

Table 2-2. Local Access Register Memory Map (continued)

Local Memory Offset (Hex)	Register	Access	Section/Page
0x0A0	DDR Local Access Window0 Base Address register (DDRLAWBAR0)	R/W	2.2.5.1.7/2-13
0x0A4	DDR Local Access Window0 Attribute register (DDRLAWAR0)	R/W	2.2.5.1.8/2-14
0x0A8–0x0BC	Reserved		
0x0C0	MBX Address Register (MBXBAR)	R/W	2.2.5.1.9/2-15
0x0C4	SRAM Address Register (SRAMBAR)	R/W	2.2.5.1.10/2-16
0x0C8	NFC Address Register (NFCBAR)	R/W	2.2.5.1.11/2-17
0x0CC–0x0FC	Reserved		

2.2.5.1 Local Access Register Description

2.2.5.1.1 Internal Memory Map Registers Base Address Register (IMMRBAR)

The internal memory map registers contain configuration, control, and status registers as well as device internal memory arrays. The internal memory map occupies a 1-Mbyte region of memory space. Its location is programmable using the internal memory map register (IMMRBAR). The default base address for the internal memory map register is 0xFF40_0000. Because IMMRBAR is at offset 0x0 from the beginning of the local access registers, IMMRBAR always points to itself. A complete list of the modules that are memory mapped by the IMMRBAR is shown in [Table 2-4](#).

2.2.5.1.2 Updating IMMRBAR

Updates to IMMRBAR that relocate the entire 1-Mbyte region of the internal memory block, requires special treatment. The effect of the update must be guaranteed to be visible by the mapping logic before an access to the new location is seen. To make sure this happens, these guidelines should be followed:

- IMMRBAR should be updated during initial configuration of the device when only one host or controller has access to the device
- During system initialization, immediately after the release of reset, system software should set IMMRBAR to the desired final location before enabling other I/O devices to access the device. A copy of the IMMRBAR value should be written to Special Purpose Register SPR311 (MBAR). Updating SPR311 is not automatic. When software changes IMMRBAR, SPR311 should be updated by software at the same time.

The Internal Memory Map Registers' Base Address register is shown in [Figure 2-1](#).

Offset 0x00 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BASE_ADDR												0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 2-1. Internal Memory Map Registers' Base Address Register (IMMRBAR)

Table 2-3. IMMRBAR Field Descriptions

Field	Description
BASE_ADDR	Identifies the 12 most-significant address bits of the base of the 4 Mbyte internal memory window.

Table 2-4. MPC5121e Memory Map

Address (OFFSET FROM IMMRBAR)	Use
00000 – 001FF	System configuration (XLBMEN)
00200 – 008FF	Reserved
00900 – 009FF	Software watchdog timer (WDT)
00A00 – 00AFF	Real time clock (RTC)
00B00 – 00BFF	General purpose timer (GPT)
00C00 – 00CFF	Integrated programmable interrupt controller (IPIC)
00D00 – 00DFF	CSB arbiter
00E00 – 00EFF	Reset module (RESET)
00F00 – 00FFF	Clock module (CLOCK)
01000 – 010FF	Power management control (PMC)
01100 – 011FF	General Purpose I/O (GPIO)
01200 – 012FF	Reserved
01300 – 0137F	MSCAN 1

Table 2-4. MPC5121e Memory Map (continued)

Address (OFFSET FROM IMMRBAR)	Use
01380 – 013FF	MSCAN 2
01400 – 014FF	Byte data link controller (BDLC)
01500 – 015FF	Secure digital host controller (SDHC)
01600 – 016FF	Sony/Philips digital interface (SPDIF)
01700 – 0171F	Inter-integrated circuit (I ² C) 1
01720 – 0173F	Inter-integrated circuit (I ² C) 2
01740 – 017FF	Inter-integrated circuit (I ² C) 3
01800 – 01FFF	Reserved
02000 – 020FF	AXE
02100 – 021FF	Display Interface Unit (DIU)
02200 – 022FF	Clock Frequency Measurement (CFM)
02300 – 0237F	MSCAN 3
02380 – 023FF	MSCAN 4
02400 – 027FF	VIU
02800 – 02FFF	Fast Ethernet Controller (FEC)
03000 – 035FF	USB ULPI
03600 – 03FFF	Reserved
04000 – 045FF	USB UTMI
07000 – 07FFF	Reserved
08000 – 082FF	PCI DMA
08300 – 0837F	PCI configuration
08380 – 083FF	Reserved
08400 – 084FF	PCI I/O Sequencer (IOS)
08500 – 085FF	PCI controller
08600 – 08FFF	Reserved
09000 – 09FFF	Multi-port DRAM controller (MDDRC)
0A000 – 0AFFF	IO control
0B000 – 0BFFF	IC identification module (IIM)
0C000 – 0FFFF	Reserved
10000 – 101FF	LocalPlus controller (LPC)
10200–102FF	Parallel ATA (PATA)
10300 – 10FFF	Reserved
11000 – 110FF	PSC0
11100 – 111FF	PSC1

Table 2-4. MPC5121e Memory Map (continued)

Address (OFFSET FROM IMMRBAR)	Use
11200 – 112FF	PSC2
11300 – 113FF	PSC3
11400 – 114FF	PSC4
11500 – 115FF	PSC5
11600 – 116FF	PSC6
11700 – 117FF	PSC7
11800 – 118FF	PSC8
11900 – 119FF	PSC9
11A00 – 11AFF	PSC10
11B00 – 11BFF	PSC11
11F00 – 11FFF	SFIFO for PSC 0–11
12000 – 13FFF	Reserved
14000 – 157FF	DMA
15800 – 1FFFF	Reserved
20000 – 21FFF	SATA
22000 – FFFFF	Reserved

2.2.5.1.3 LocalPlus Boot/CS0-7 Access Window Registers (LPBAW/LPCSxAW)

The LocalPlus Access Window Registers (LPBAW/LPCSxAW) are shown in [Figure 2-2](#).

Offset 0x20,0x24,0x28,0x2C,0x30,0x34,0x38,0x3C,0x40

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	START_ADDR															
W	START_ADDR															
Reset ¹	See Table 2-6															
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	STOP_ADDR															
W	STOP_ADDR															
Reset	See Table 2-6															

1. The LPBAW reset value depends on the reset configuration word high values. See [Section 2.2.5.1.4](#), “LPBAW[START_ADDR] and LPBAW[STOP_ADDR] Reset Value” for detailed description.

Figure 2-2. LocalPlus Boot/CS0–7 Access Window Registers (LPBAW/LPCSxAW)

Table 2-5. LPBAW/LPCSxAW Field Descriptions

Field	Description
START_ADDR	Any access on an address between Start and Stop Address enables the corresponding chip select. The START_ADDR is for the address comparison extended with 0x0000. The STOP_ADDR is for the address comparison extended with 0xFFFF. This means the minimum address size is 64 Kbytes. If the START_ADDR and STOP_ADDR are set to 0xA000, the access window goes from 0xA000_0000 to 0xA000_FFFF. Note: CS Boot and CS0 have the same physical CS pin.
STOP_ADDR	

2.2.5.1.4 LPBAW[START_ADDR] and LPBAW[STOP_ADDR] Reset Value

The Power Architecture core may fetch its boot vector from a local bus peripheral device. For this purpose, LPBAW[START_ADDR] and LPBAW[STOP_ADDR] reset values are set according to the value of BMS and ROM_LOC bits in the reset configuration word high register.

Table 2-6 defines the reset value LPBAW[START_ADDR] and LPBAW[STOP_ADDR].

Table 2-6. LPBAW[START_ADDR] and LPBAW[STOP_ADDR] Reset Value

ROM_LOC	BMS	START_ADDR Reset Value	STOP_ADDR Reset Value
00	0	0x0000	0x007F
00	1	0xFF80	0xFFFF
01 or 1X	X	0x0100	0x0100

NOTE

BMS is bit 26 of the reset configuration word high register (RCWHR). Its initial value is set by the state of the EMB_AD05 pin at the release of PORESET. ROM_LOC is set by bits 22 and 21 of the reset configuration word high register (RCWHR). The initial values of these bits are set by the states of the EMB_AD0 and EMB_AD1 pins at the release of PORESET.

See Figure 2-10.

2.2.5.1.5 PCI Local Access Window n Base Address Registers (PCILAWBAR0 – PCILAWBAR2)

The PCI Local Access Window n Base Address Registers (PCILAWBAR0–PCILAWBAR2) are shown in Figure 2-3.

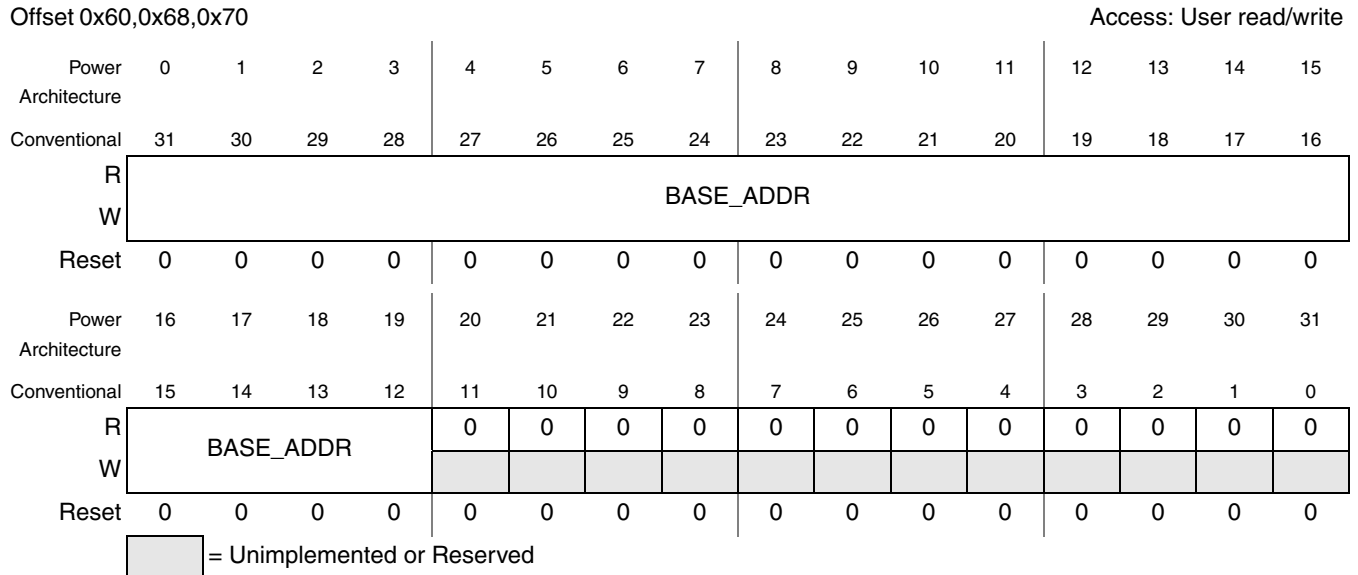


Figure 2-3. PCI Local Access Window n Base Address Registers (PCILAWBAR0 – PCILAWBAR2)

Table 2-7. PCILAWBAR0 – PCILAWBAR2 Field Descriptions

Field	Description
BASE_ADDR	Identifies the 20 most-significant address bits of the base address of local access window n. The specified base address should be aligned to the window size, as defined by PCILAWARn[SIZE].

2.2.5.1.6 PCI Local Access Window n Attributes Registers (PCILAWAR0 – PCILAWAR2)

The PCI Local Access Window n Attributes Registers (PCILAWAR0-PCILAWAR2) are shown in Figure 2-4.

Offset 0x64,0x6C,0x74

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	SIZE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 2-4. PCI Local Access Window n Attributes Registers (PCILAWAR0 – PCILAWAR2)

Table 2-8. PCILAWAR0 – PCILAWAR2 Field Descriptions

Field	Description
EN	0 The PCI local access window n is disabled. 1 The PCI local access window n is enabled and other PCILAWARn and PCILAWBARn fields combine to identify an address range for this window.
SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

2.2.5.1.7 DDR Local Access Window Base Address Register (DDRLAWBAR)

Figure 2-5 shows the DDR Local Access Window Base Address Register (DDRLAWBAR).

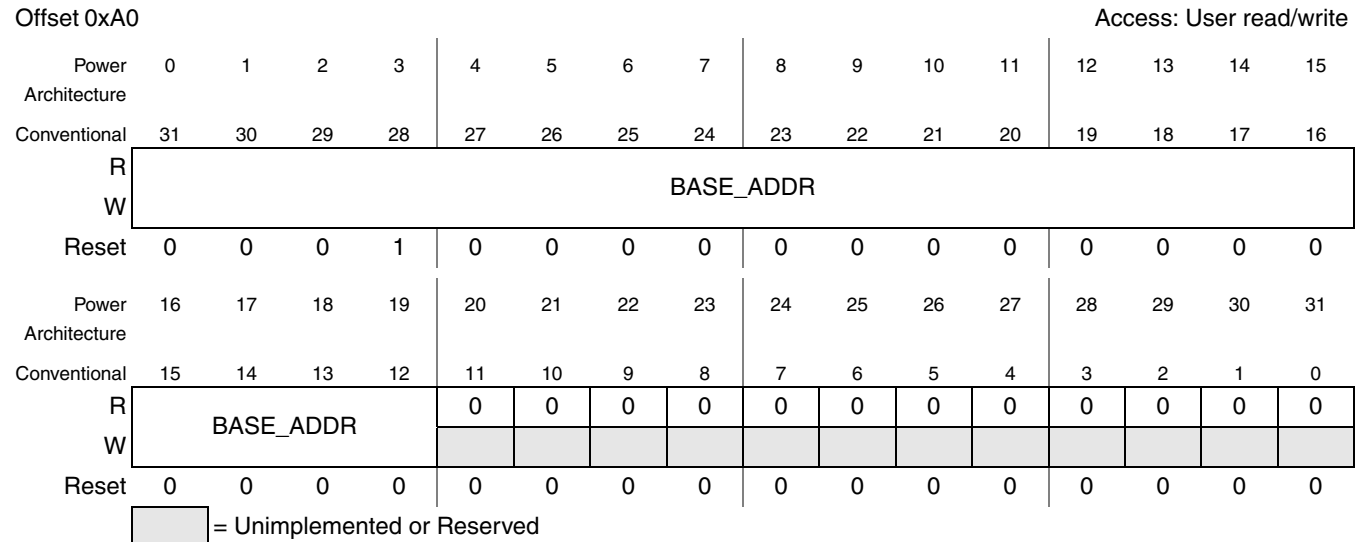


Figure 2-5. DDR Local Access Window Base Address Register (DDRLAWBAR)

Table 2-9. DDRLAWBAR0 Field Description

Field	Description
BASE_ADDR	Identifies the 20 most-significant address bits of the base address of local access window. The specified base address should be aligned to the window size, as defined by DDRLAWAR[SIZE].

2.2.5.1.8 DDR Local Access Window Attributes Register (DDRLAWAR)

Figure 2-6 shows the DDR Local Access Window Attributes Register (DDRLAWAR).

Offset 0xA4

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	SIZE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1


 = Unimplemented or Reserved

Figure 2-6. DDR Local Access Window Attributes Register (DDRLAWAR)

Table 2-10. DDRLAWAR Field Description

Field	Description
SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes.
000000–011000	Reserved. Window is undefined.
011001	64 Mbytes
011010	128 Mbytes
011011	256 Mbytes
.....	
.....	$2^{(SIZE+1)}$ bytes
.....	
011110	2 Gbytes
011111–111111	Reserved. Window is undefined.

2.2.5.1.9 MBX Base Address Register (MBXBAR)

Figure 2-7 shows the MBX Base Address Register (MBXBAR).

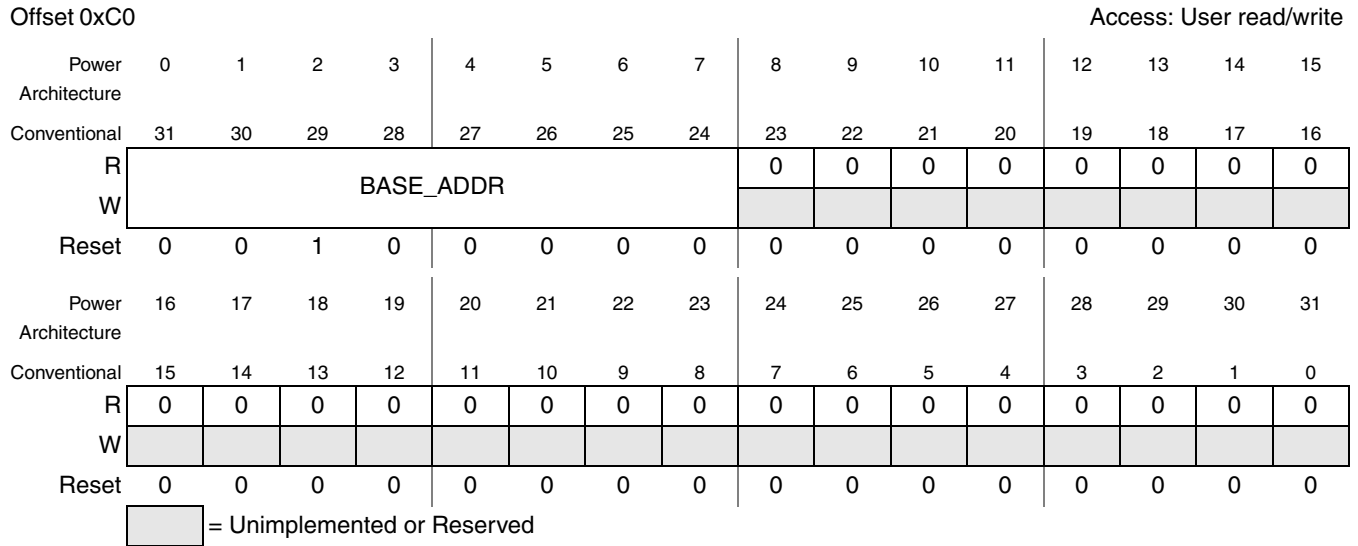


Figure 2-7. MBX Base Address Register (MBXBAR)

Table 2-11. MBXBAR Field Descriptions

Field	Description
BASE_ADDR	Identifies the 8 most-significant address bits of the base address of the 16 Mbyte MBX memory window.

2.2.5.1.10 SRAM Base Address Register (SRAMBAR)

Figure 2-8 shows the SRAM Base Address Register (SRAMBAR).

NOTE

Although the SDRAM window size is 256 Kbytes, the MPC5121e SDRAM size is 128 Kbytes.

Offset 0xC4												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BASE_ADDR														0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 2-8. SRAM Base Address Register (SRAMBAR)

Table 2-12. SRAMBAR Field Descriptions

Field	Description
BASE_ADDR	Identifies the 14 most-significant address bits of the base address of the 256-Kbyte SRAM memory window.

2.2.5.1.11 NFC Base Address Register (NFCBAR)

Figure 2-9 shows the NFC Base Address Register (NFCBAR).

Offset 0xC8												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BASE_ADDR												0	0	0	0
W																
Reset	See Table 2-14												0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The NFCBAR[BASE_ADDR] reset value depends on the reset configuration word. See Section 2.2.5.1.12, “NFCBAR[BASE_ADDR] Reset Value” for detailed description.


 = Unimplemented or Reserved

Figure 2-9. NFC Base Address Registers (NFCBAR)

Table 2-13. NFCBAR Field Descriptions

Field	Description
BASE_ADDR	Identifies the 12 most-significant address bits of the base address of the 1 Mbyte NFC memory window.

2.2.5.1.12 NFCBAR[BASE_ADDR] Reset Value

The Power Architecture core may use a NAND flash device to fetch its boot vector. For this purpose, NFCBAR[BASE_ADDR] reset value is set according to the value set in the reset configuration word high BMS and ROM_LOC fields.

Table 2-14 defines the reset value NFCBAR[BASE_ADDR].

Table 2-14. NFCBAR[BASE_ADDR] Reset Value

ROM_LOC	BMS	BASE_ADDR Reset Value
X1	0	0x000
X1	1	0xFFF
X0	X	0x400

NOTE

BMS is bit 26 of the reset configuration word high register (RCWHR). Its initial value is set by the state of the EMB_AD05 pin at the release of $\overline{\text{PORESET}}$. ROM_LOC is set by bits 22 and 21 of the reset configuration word high register (RCWHR). The initial values of these bits are set by the states of the EMB_AD0 and EMB_AD1 pins at the release of $\overline{\text{PORESET}}$.

See [Table 2-15](#) for initial memory map values based on the EMBAD0/1 and BMS bits.

Table 2-15. Memory Map Values based on EMBAD0/1 and BMS

ROM_LOC	BMS	BOOT_START	BOOT_STOP	NFC_BASE_ADDR	NFC_STOP_ADDR
00	0	0x0000 0000	0x007F FFFF	0x4000 0000	0x400F FFFF
00	1	0xFF80 0000	0xFFFFF FFFF	0x4000 0000	0x400F FFFF
01	0	0x0100 0000	0x0100 FFFF	0x0000 0000	0x000F FFFF
01	1	0x0100 0000	0x0100 FFFF	0xFFF0 0000	0xFFFF FFFF
10	0	0x0100 0000	0x0100 FFFF	0x4000 0000	0x400F FFFF
10	1	0x0100 0000	0x0100 FFFF	0x4000 0000	0x400F FFFF
11	0	0x0100 0000	0x0100 FFFF	0x0000 0000	0x000F FFFF
11	1	0x 0100 0000	0x0100 FFFF	0xFFF0 0000	0xFFFF FFFF

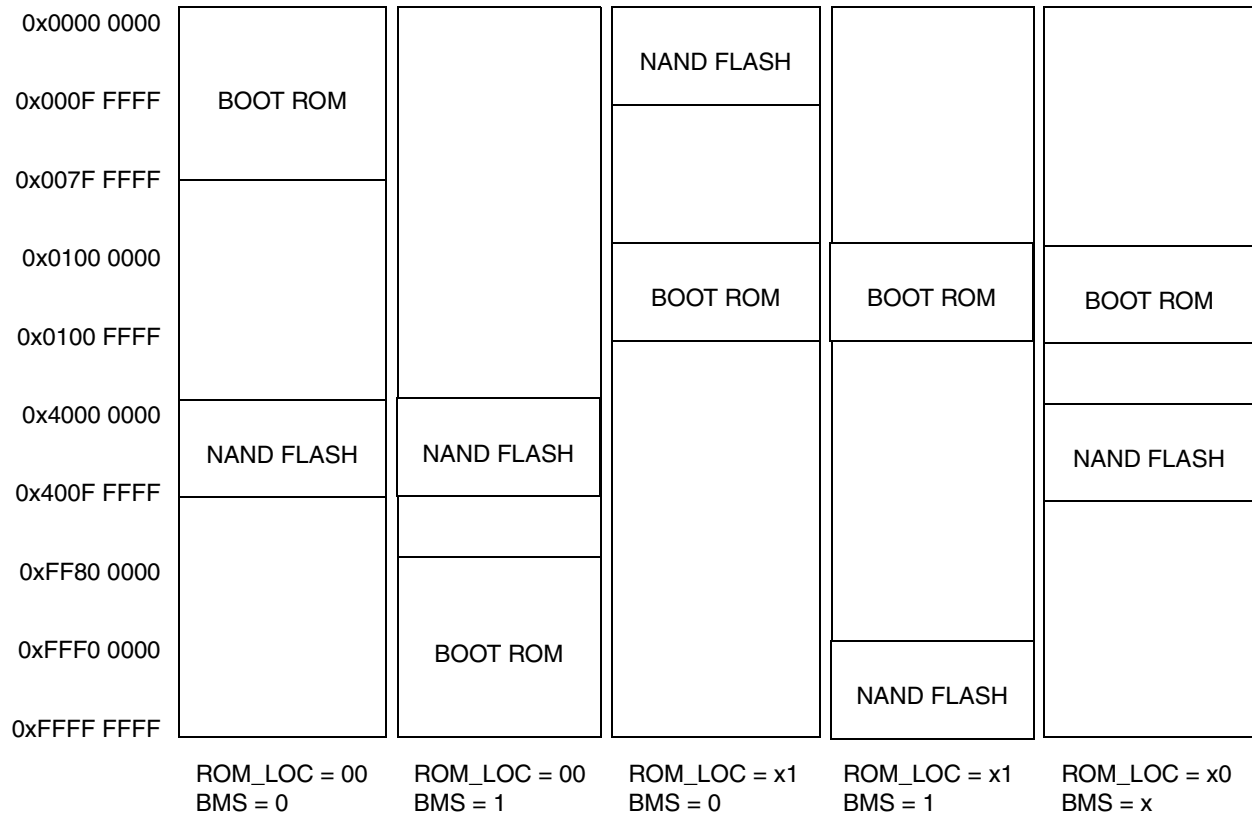


Figure 2-10. Initial Memory Map Configurations Immediately After the Release of $\overline{\text{PORESET}}$

2.2.6 Precedence of Local Access Windows

In general, memory windows should not overlap. In case of overlap, the lower numbered memory window takes precedence over the higher numbered memory window.

2.2.7 Configuring Local Access Windows

After a local access window is enabled, it should not be modified while any device in the system may be using the window. Neither should a new window be used until the effect of the write to the window is visible to all blocks that use the window. This can be guaranteed by completing a read of the last local access window configuration register before enabling any other devices to use the window. For instance, if LPC local access windows 1–3 are being configured in order during the initialization process, the last write (to LPCS2AW) should be followed by a read of LPCS2AW before any devices try to use any of these windows. If the configuration is being done by the local Power Architecture processor, the read of LPCS2AW should be followed by an isync instruction.

2.2.8 Distinguishing Local Access Windows from Other Mapping Functions

It is important to distinguish between the mapping function performed by the local access windows and the additional mapping functions that happen at the target interface. The local access windows define how a transaction is routed through the MPC5121e internal interconnects from the transactions source to its target. After the transaction has arrived at its target interface, that interface controller may do additional mapping. For instance, the DDR SDRAM controller has chip select registers that map a memory request to a particular external device. Similarly, the local bus controller has base registers that perform a similar function. The PCI interface has outbound address translation units that map the local address into an external address space.

These other mapping functions are configured by programming the configuration, control, and status registers of the individual interfaces. There is no need to have a one-to-one correspondence between local access windows and chip select regions on outbound windows. A single local access window can be further decoded to any number of chip selects or to any number of outbound windows at the target interface.

2.2.9 Outbound Address Translation and Mapping Windows

Outbound address translation and mapping refers to the translation of addresses from the local 32-bit address space to the external address space and attributes of a particular I/O interface.

The PCI controller has three outbound windows.

2.2.10 Inbound Address Translation and Mapping Windows

Inbound address translation and mapping refers to the translation of an address from the external address space of an I/O interface (such as PCI address space) to the local address space understood by the internal interfaces of the MPC5121e. It also refers to the mapping of transactions to a particular target interface and the assignment of transaction attributes. The PCI controller has inbound address translation unit.

2.2.11 PCI Inbound Windows

The PCI controller has three general inbound windows for memory mapped configuration accesses (PIMMR). These windows have a one-to-one correspondence with the base address registers in the PCI programming model. Updating one automatically updates the other. There is no default inbound window. If a PCI address does not match one of the inbound windows, the MPC5121e does not respond with an assertion of `PCI_DEVSEL`. See [Section 28.3, “Memory Map and Register Definition”](#) for a detailed description of the PCI inbound windows.

2.2.12 Accessing Internal Memory from External Masters

In addition to being accessible by the Power Architecture processor, the memory window is accessible from external interfaces. This allows external masters on the I/O ports to configure the MPC5121e.

External masters do not need to know the location of the IMMR memory in the local address map. Rather, they access this region of the local memory map through a window defined by a register in the interface's programming model that is accessible to the external master from its external memory map.

The PCI base address for accessing the local IMMR memory is selectable through the PCI internal memory map register (PIMMR), at offset 0x10, described in [Section 28.3, “Memory Map and Register Definition”](#).

2.3 System Configuration

Some general information and configuration options which affect the system behavior and performance are described in the following sections.

2.3.1 System Configuration Register Memory Map

[Table 2-16](#) shows the memory map for the system configuration registers.

Table 2-16. System Configuration Register Memory Map

Local Memory Offset (Hex)	Register	Access	Section/Page
0x100	System Part and Revision ID Register (SPRIDR)	R/W	2.3.1.1.1/2-22
0x104	System Priority Configuration Register (SPCR)	R/W	2.3.1.1.2/2-23
0x108 - 0x1FC	Reserved	—	—

2.3.1.1 System Configuration Registers

2.3.1.1.1 System Part and Revision ID Register (SPRIDR)

The System Part and Revision ID Register shown in Figure 2-11 provides information about the part and revision numbers.

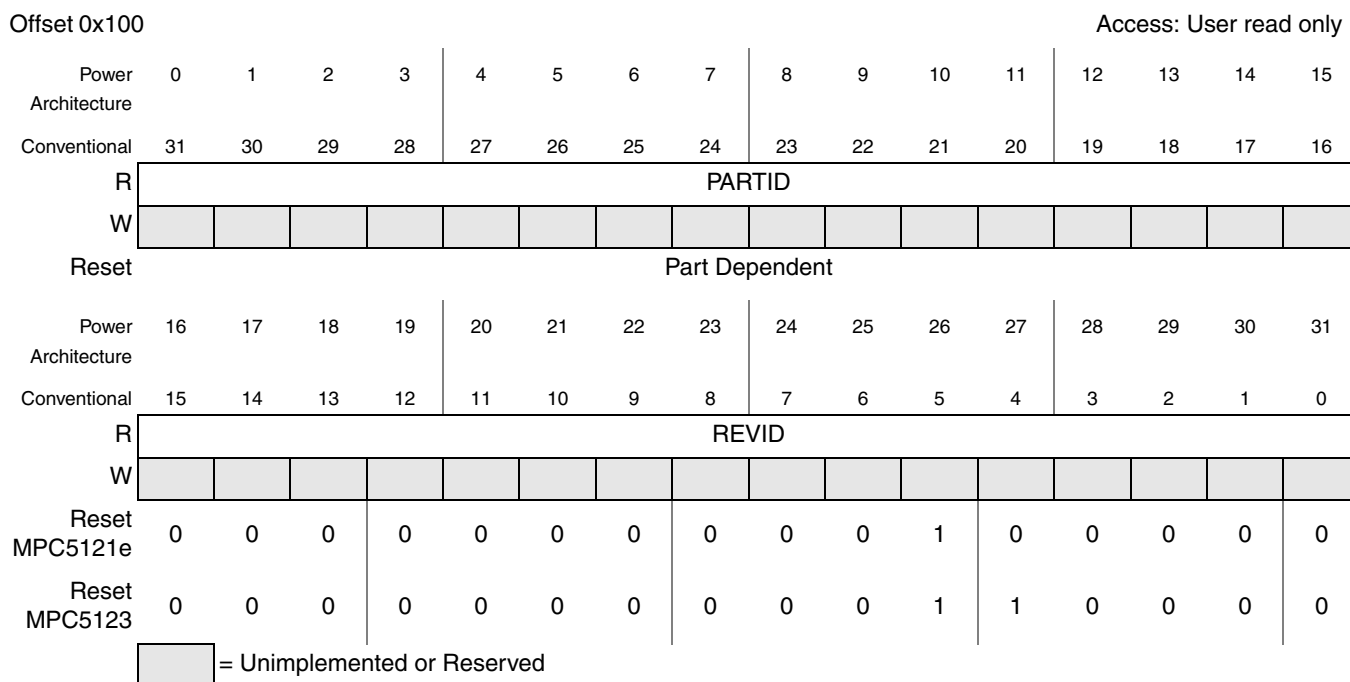


Figure 2-11. System Part and Revision ID Register (SPRIDR)

Table 2-17. SPRIDR Field Descriptions

Field	Description
PARTID	Part Identification. This read-only field is mask-programmed with a code corresponding to the part number. It is intended to help factory test and user code which is sensitive to part changes. The part number changes according to manufacturing considerations.
REVID	Revision Identification. This read-only field is mask-programmed with a code corresponding to the revision number of the part defined in PARTID field. It is intended to help factory test and user code which is sensitive to part changes. The mask number changes with each mask set change.

2.3.1.1.2 System Priority Configuration Register (SPCR)

The System Priority Configuration Register shown in Figure 2-12 controls the priority of requests for transactions on the internal system bus. This priority is considered by the system arbiter when an internal unit requests the bus.

Offset 0x104

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				PCIH												
W				PE						TBEN		COREPR				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W				SAPPR		MBX-							TEM		TEMPSEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 2-12. System Priority Configuration Register (SPCR)
(The register is repeated for reference.)

Table 2-18. SPCR Field Descriptions

Field	Description
PCIHPE	PCI Highest Priority Enable. If this bit is set, the PCI bridge is permitted to request the internal system bus with highest priority, regardless of SYCR[PCIPR] value, when it needs to complete a posted write transaction from an external PCI master. To follow PCI ordering rules specifications, the PCI bridge must flush any outstanding write transactions before it can start a new read transaction. Setting this bit allows faster flushing of the outstanding write transactions coming from the PCI bus onto the system bus and to the system targets, such as DDR SDRAM and local bus memories.
PCIPR	PCI bridge system bus request priority. The level of priority can be chosen from 4 possible levels. 00 Level 0 (Lowest Priority) 01 Level 1 10 Level 2 11 Level 3 (Highest Priority)
TBEN	Power Architecture Core time base unit enable 0 Time base unit is disabled. 1 Time base unit is enabled.
COREPR	Power Architecture Core system bus request priority. The level of priority can be chosen from 4 possible levels. 00 Level 0 (Lowest Priority) 01 Level 1 10 Level 2 11 Level 3 (Highest Priority)

System Configuration and Memory Map (XLBMEN + Mem Map)

Offset 0x104

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				PCIH												
W				PE						TBEN		COREPR				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 2-12. System Priority Configuration Register (SPCR)
(The register is repeated for reference.)

Table 2-18. SPCR Field Descriptions (continued)

Field	Description
SAPPR	SAP and TPR2MG system bus request priority. The level of priority can be chosen from 4 possible levels. 00 Level 0 (Lowest Priority) 01 Level 1 10 Level 2 11 Level 3 (Highest Priority)
MBXBSSEN	MBX Byte Swap enable bit 0 MBX Byte Swap is disabled. 1 MBX Byte Swap is enabled.
TEMPPD	Temperature Sensor power down 0 Temperature Sensor power is enabled 1 Temperature Sensor power is disabled
TEMPSEL	Temperature Sensor select trip point for tempflt_lo 000 105 °C 001 95 °C 010 85 °C 011 75 °C 100 65 °C 101 55 °C 110 45 °C 111 35 °C

Chapter 3

Signal Descriptions

3.1 Introduction

This chapter describes the MPC5121e external signals. It is organized into the following sections:

- Overview of signals and cross references for signals that serve multiple functions, including two lists: one ordered by functional block and one alphabetical
- List of reset configuration signals
- List of output signal states during reset

NOTE

A bar over a signal name indicates that the signal is active low, such as $\overline{\text{IRQ0}}$ (interrupt input). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as MODT (DDR2 on-die-termination output), are referred to as asserted when they are high and negated when they are low.

3.1.1 Signals Overview

The MPC5121e signals are grouped as follows:

- DDR memory interface signals
- PCI interface signals
- PSC interface signals
- I²C interface signals
- LPC interface signals
- PATA interface signals
- NFC interface signals
- EMB interface signals
- SATA interface signals
- USB Phy interface signals
- CAN interface signals
- J1850 interface signals
- SPDIF interface signals
- JTAG, test, system control signals
- Clock signals

Figure 3-1 shows the primary external signals of the MPC5121e and how the signals are grouped. Refer to the *MPC5121e Data Sheet* at www.freescale.com for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications.

Functionality, which is not a primary function (DIU, FEC, USB ULPI), is not shown in Figure 3-1. These functions are multiplexed. Multiplexing is shown in Table 3-1.

Individual chapters of this document provide details for each signal, describing each signal's behavior when asserted and negated and when the signal is an input or an output. Power signals are described in the Hardware Specification.

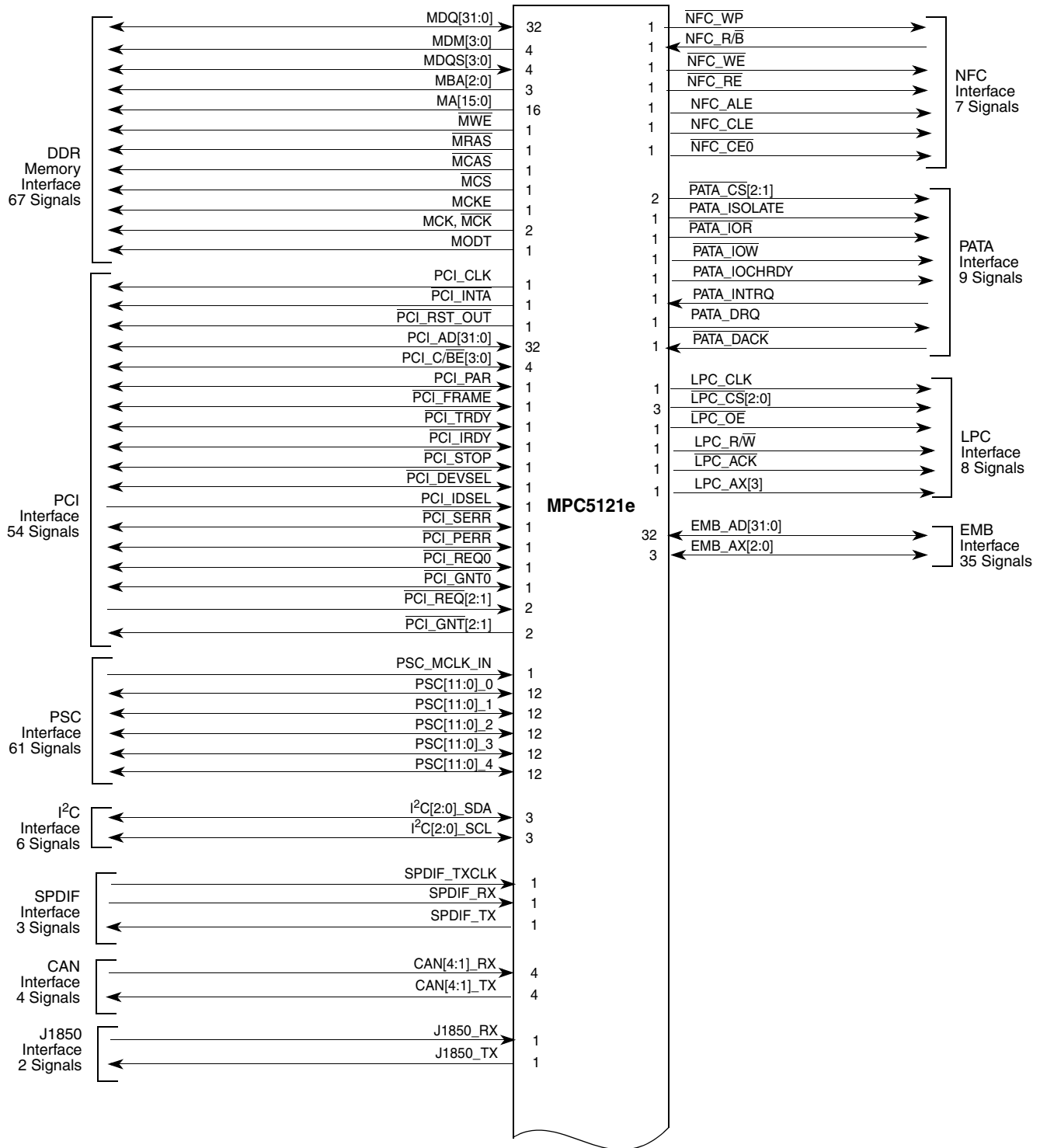


Figure 3-1. MPC5121e Signal Groupings

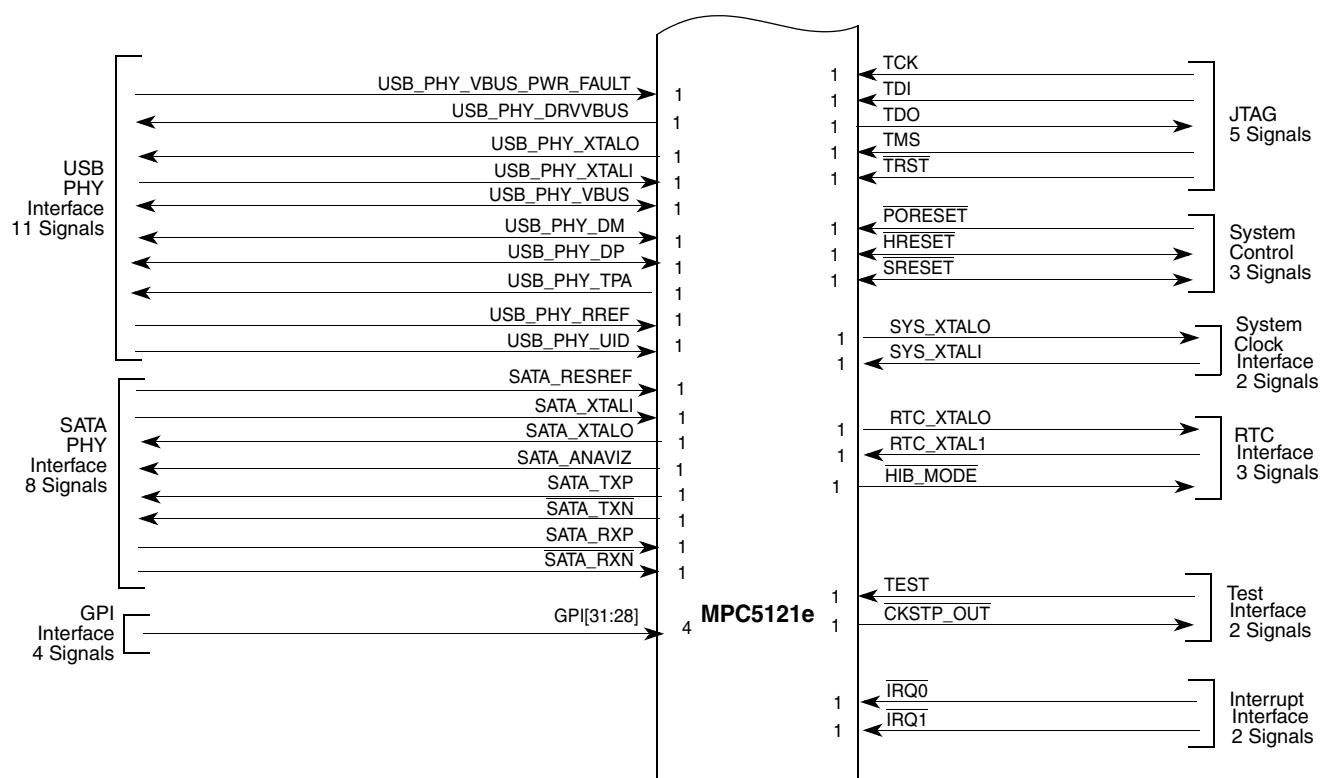


Figure 3-1. MPC5121e Signal Groupings (continued)

The following tables provide summaries of signal functions. [Table 3-1](#) provides a summary of the signals grouped by function, and [Table 3-2](#) provides a summary of the signals grouped alphabetically. These tables detail the signal name, interface, alternate functions, number of signals, and whether the signal is an input, output, or bidirectional.

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 1 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
MDQ0	—	—	—
MDQ1	—	—	—
MDQ2	—	—	—
MDQ3	—	—	—
MDQ4	—	—	—
MDQ5	—	—	—
MDQ6	—	—	—
MDQ7	—	—	—
MDQ8	—	—	—
MDQ9	—	—	—
MDQ10	—	—	—
MDQ11	—	—	—

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 2 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
MDQ[12:15]	—	—	—
MDQ[16:23]	—	—	GPIO[0:7]
MDQ[24:31]	—	—	GPIO[16:23]
MDM[0:1]	—	—	—
MDM[2:3]	—	—	GPIO[24:25]
MDQS[0:1]	—	—	—
MDQS[2:3]	—	—	GPIO[26:27]
MBA[0:2]	—	—	—
MA[0:15]	—	—	—
$\overline{\text{MWE}}$	—	—	—
$\overline{\text{MRAS}}$	—	—	—
$\overline{\text{MCAS}}$	—	—	—
$\overline{\text{MCS}}$	—	—	—
MCK	—	—	—
$\overline{\text{MCK}}$	—	—	—
MCKE	—	—	—
MODT	—	—	—
MVREF	—	—	—
MVTT[0:3]	—	—	—
LPC_CLK	TPA	CKSTP_IN	—
$\overline{\text{LPC_OE}}$	—	—	—
LPC_R/W	—	—	—
$\overline{\text{LPC_CS0}}$	—	—	GPIO25
$\overline{\text{LPC_CS1}}$	SPDIF_TXCLK	—	GPIO7
$\overline{\text{LPC_CS2}}$	$\overline{\text{NFC_CE_1}}$	—	GPIO0
$\overline{\text{LPC_ACK}}$	$\overline{\text{LPC_CS7}}$	—	GPIO24
LPC_AX03	—	—	GPIO1
EMB_AX02	—	—	GPIO2
EMB_AX01	—	—	GPIO3
EMB_AX00	—	—	—
—	—	EMB_AD[31:0]	—
$\overline{\text{PATA_CE1}}$	$\overline{\text{LPC_CS4}}$	—	GPIO9
$\overline{\text{PATA_CE2}}$	$\overline{\text{LPC_CS5}}$	—	GPIO10
PATA_ISOLATE	CAN3_TX	—	GPIO11
$\overline{\text{PATA_IOR}}$	SDHC_CLK	—	GPIO12
$\overline{\text{PATA_IOW}}$	SDHC_CMD	LPC_AX08	GPIO13

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 3 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
PATA_IOCHRDY	SDHC_D0	LPC_AX07	GPIO14
PATA_INTRQ	SDHC_D1_IRQ	LPC_AX06	GPIO15
PATA_DRQ	SDHC_D2	LPC_AX05	GPIO16
PATA_DACK	SDHC_D3_CD	LPC_AX04	GPIO17
NFC_CE_0	LPC_CS3	PSC_MCLK_IN	GPIO26
NFC_WP	SDHC_CLK	LPC_AX09	GPIO18
NFC_R/B	SDHC_CMD	LPC_AX08	GPIO19
NFC_ALE	SDHC_D0	LPC_AX07	GPIO20
NFC_CLE	SDHC_D1_IRQ	LPC_AX06	GPIO21
NFC_WE	SDHC_D2	LPC_AX05	GPIO22
NFC_RE	SDHC_D3_CD	LPC_AX04	GPIO23
I2C0_SCL	—	—	GPIO7/GPT7
I2C0_SDA	—	—	GPIO1
I2C1_SCL	—	SPDIF_TX	GPIO2
I2C1_SDA	—	SPDIF_RX	GPIO3
I2C2_SCL	—	CAN4_TX	GPIO4
I2C2_SDA	—	CAN4_RX	GPIO5
IRQ0	—	CAN3_TX	GPIO4/GPT4
IRQ1	SPDIF_TXCLK	CAN3_RX	GPIO5/GPT5
CAN1_TX	—	—	GPIO6
CAN1_RX	—	—	—
CAN2_TX	—	—	GPIO8
CAN2_RX	—	—	—
J1850_TX	—	GPIO4	CAN4_TX
J1850_RX	—	LPC_CS6	CAN4_RX
SPDIF_TXCLK	FEC_RX_DV	DIU_CLK	GPIO26
SPDIF_TX	FEC_TX_ER	DIU_VSYNC	GPIO27
SPDIF_RX	FEC_CRS	DIU_HSYNC	GPIO0
PCI_GNT2	—	DIU_LD21	GPIO8
PCI_REQ2	—	DIU_LD20	GPIO9
PCI_GNT1	—	DIU_LD19	GPIO10
PCI_REQ1	—	DIU_LD18	GPIO11
PCI_INTA	—	DIU_LD15	GPIO14
PCI_CLK	—	DIU_LD14	GPIO15
PCI_RST_OUT	—	DIU_LD13	GPIO16
PCI_GNT0	—	DIU_LD12	GPIO12

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 4 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
PCI_REQ0	—	DIU_LD11	GPIO13
PCI_FRAME	—	DIU_LD10	GPIO17
PCI_IDSEL	—	DIU_LD07	GPIO18
PCI_DEVSEL	—	DIU_LD06	GPIO19
PCI_IRDY	USB0_DATA7	DIU_LD05	GPIO20
PCI_TRDY	USB0_DATA6	DIU_LD04	GPIO21
PCI_C/BE0	USB0_DATA5	DIU_LD03	GPIO4
PCI_C/BE1	USB0_DATA4	DIU_LD02	GPIO5
PCI_C/BE2	USB0_DATA3	DIU_LD00	GPIO6
PCI_C/BE3	USB0_DATA2	DIU_LD01	GPIO7
PCI_STOP	USB0_DATA1	DIU_LD08	GPIO22
PCI_PAR	USB0_DATA0	DIU_LD09	GPIO23
PCI_PERR	USB0_STOP	DIU_LD16	GPIO24
PCI_SERR	USB0_NEXT	DIU_LD17	GPIO25
PCI_AD31	USB0_CLK	DIU_LD22	GPIO0
PCI_AD30	USB0_DIR	DIU_LD23	GPIO1
PCI_AD[29:22]	—	USB1_DATA[7:0]	GPIO[2:9]
PCI_AD21	—	USB1_STOP	GPIO10
PCI_AD20	—	USB1_NEXT	GPIO11
PCI_AD19	—	USB1_CLK	GPIO12
PCI_AD18	—	USB1_DIR	GPIO13
PCI_AD17	VIU_DATA0	FEC_TXD_3	GPIO14
PCI_AD16	VIU_DATA1	FEC_TXD_2	GPIO15
PCI_AD15	VIU_DATA2	FEC_TXD_1	GPIO16
PCI_AD14	—	FEC_TXD_0	GPIO17
PCI_AD13	VIU_DATA3	FEC_RXD_3	GPIO18
PCI_AD12	VIU_DATA4	FEC_RXD_2	GPIO19
PCI_AD11	VIU_DATA5	FEC_RXD_1	GPIO20
PCI_AD10	—	FEC_RXD_0	GPIO21
PCI_AD09	—	FEC_RX_CLK	GPIO22
PCI_AD08	—	FEC_TX_CLK	GPIO23
PCI_AD07	VIU_DATA7	FEC_RX_ER	GPIO24
PCI_AD06	—	FEC_RX_DV	GPIO25
PCI_AD05	—	FEC_TX_EN	GPIO26
PCI_AD04	VIU_PIX_CLK	FEC_TX_ER	GPIO27
PCI_AD03	VIU_DATA6	FEC_CRD	GPIO0

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 5 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
PCI_AD02	VIU_DATA8	FEC_MDC	GPIO1
PCI_AD01	VIU_DATA9	FEC_MDIO	GPIO2
PCI_AD00	—	FEC_COL	GPIO3
PSC_MCLK_IN	—	DIU_DE	GPIO6/GPT6
PSC0_0	FEC_COL	USB0_DATA7	GPIO8
PSC0_1	FEC_TX_EN	USB0_DATA6	GPIO9
PSC0_2	FEC_TX_CLK	USB0_DATA5	GPIO10
PSC0_3	FEC_TXD_0	USB0_DATA4	GPIO11
PSC0_4	FEC_TXD_1	USB0_DATA3	GPIO0/GPT0
PSC1_0	FEC_TXD_2	USB0_DATA2	GPIO12
PSC1_1	FEC_TXD_3	USB0_DATA1	GPIO13
PSC1_2	FEC_MDC	USB0_DATA0	GPIO14
PSC1_3	FEC_RX_ER	USB0_STOP	GPIO15
PSC1_4	FEC_RXD_3	USB0_NEXT	GPIO1/GPT1
PSC2_0	FEC_RXD_2	USB0_CLK	GPIO16
PSC2_1	FEC_RXD_1	USB0_DIR	GPIO17
PSC2_2	FEC_RXD_0	—	GPIO18
PSC2_3	FEC_MDIO	—	GPIO19
PSC2_4	FEC_RX_CLK	—	GPIO2/GPT2
PSC3_[0:3]	USB1_DATA[0:3]	—	GPIO[20:23]
PSC3_4	$\overline{\text{LPC_CS6}}$	VIU_PIX_CLK	GPIO3/GPT3
PSC4_[0:3]	USB1_DATA[4:7]	VIU_DATA[0:3]	GPIO[24:27]
PSC4_4	$\overline{\text{NFC_CE2}}$	VIU_DATA4	GPIO4/GPT4
PSC5_0	USB1_CLK	VIU_DATA5	GPIO8
PSC5_1	USB1_NEXT	VIU_DATA6	GPIO9
PSC5_2	USB1_STOP	VIU_DATA7	GPIO10
PSC5_3	USB1_DIR	VIU_DATA8	GPIO11
PSC5_4	$\overline{\text{NFC_CE3}}$	VIU_DATA9	GPIO5/GPT5
PSC6_0	LPC_TSIZ1	DIU_CLK	GPIO12
PSC6_1	LPC_TSIZ2	DIU_HSYNC	GPIO13
PSC6_2	—	—	GPIO14
PSC6_3	—	—	GPIO15
PSC6_4	LPC_TS	DIU_VSYNC	GPIO6/GPT6
PSC7_0	SDHC_CMD	DIU_LD23	GPIO16
PSC7_1	SDHC_D0	DIU_LD22	GPIO17
PSC7_2	SDHC_D1_IRQ	DIU_LD17	GPIO18

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 6 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
PSC7_3	SDHC_D2	DIU_LD16	GPIO19
PSC7_4	SDHC_D3_CD	DIU_LD09	GPIO7/GPT7
PSC8_0	—	DIU_LD08	GPIO20
PSC8_1	—	DIU_LD01	GPIO21
PSC8_2	—	DIU_LD00	GPIO22
PSC8_3	—	DIU_LD02	GPIO23
PSC8_4	SDHC_CLK	DIU_LD03	GPIO0/GPT0
PSC9_[0:3]	—	DIU_LD[04:7]	GPIO[24:27]
PSC9_4	—	DIU_LD10	GPIO1/GPT1
PSC10_[0:3]	—	DIU_LD[11:14]	GPIO[8:11]
PSC10_4	—	DIU_LD15	GPIO2/GPT2
PSC11_[0:3]	—	DIU_LD[18:21]	GPIO[12:15]
PSC11_4	—	DIU_DE	GPIO3/GPT3
TCK	—	—	—
TDI	—	—	—
TDO	—	—	—
TMS	—	—	—
$\overline{\text{TRST}}$	—	—	—
$\overline{\text{PORESET}}$	—	—	—
$\overline{\text{HRESET}}$	—	—	—
$\overline{\text{SRESET}}$	—	—	—
TEST	—	—	—
$\overline{\text{CKSTP_OUT}}$	TPA	—	—
GPIO28	—	—	—
GPIO29	—	—	—
GPIO30	—	—	—
GPIO31	—	—	—
SYS_XTALO	—	—	—
SYS_XTALI	—	—	—
RTC_XTALO	—	—	—
RTC_XTALI	—	—	—
$\overline{\text{HIB_MODE}}$	—	—	—
USB_PHY_VBUS	—	—	—
USB_PHY_RREF	—	—	—
USB_PHY_UID	—	—	—
USB_PHY_TPA	—	—	—

Table 3-1. MPC5121e Signal Reference by Functional Block (Sheet 7 of 7)

Pin Name Function 1	Function 2	Function 3	Function 4
USB_PHY_XTALO	—	—	—
USB_PHY_XTALI	—	—	—
USB_PHY_DM	—	—	—
USB_PHY_DP	—	—	—
USB_PHY_VBUS_ PWR_FAULT	—	—	—
USB_PHY_DRVVBUS	—	—	—
SATA_XTALI	—	—	—
SATA_XTALO	—	—	—
SATA_ANAVIZ	—	—	—
SATA_RESREF	—	—	—
SATA_TXP	—	—	—
SATA_TXN	—	—	—
SATA_RXP	—	—	—
SATA_RXN	—	—	—

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 1 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
—	—	—	EMB_AD[31:0]	—
CAN1_RX	C19	—	—	—
CAN1_TX	A18	—	—	GPIO6
CAN2_RX	B19	—	—	—
CAN2_TX	E16	—	—	GPIO8
CKSTP_OUT	Y26	TPA	—	—
EMB_AX00	W2	—	—	—
EMB_AX01	V5	—	—	GPIO3
EMB_AX02	W3	—	—	GPIO2
GPIO28	A19	—	—	—
GPIO29	E17	—	—	—
GPIO30	C18	—	—	—
GPIO31	B18	—	—	—
HIB_MODE	D18	—	—	—
HRESET	W24	—	—	—
I2C0_SCL	AC23	—	—	GPIO7/GPT7
I2C0_SDA	AD26	—	—	GPIO1

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 2 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
I2C1_SCL	AB22	—	SPDIF_TX	GPIO2
I2C1_SDA	AB23	—	SPDIF_RX	GPIO3
I2C2_SCL	AC25	—	CAN4_TX	GPIO4
I2C2_SDA	AA22	—	CAN4_RX	GPIO5
$\overline{\text{IRQ0}}$	AC26	—	CAN3_TX	GPIO4/GPT4
$\overline{\text{IRQ1}}$	AB25	SPDIF_TXCLK	CAN3_RX	GPIO5/GPT5
J1850_RX	AA24	—	$\overline{\text{LPC_CS6}}$	CAN4_RX
J1850_TX	Y22	—	GPIO4	CAN4_TX
$\overline{\text{LPC_ACK}}$	AA2	$\overline{\text{LPC_CS7}}$	—	GPIO24
LPC_AX03	W4	—	—	GPIO1
LPC_CLK	AA4	TPA	CKSTP_IN	—
$\overline{\text{LPC_CS0}}$	W5	—	—	GPIO25
$\overline{\text{LPC_CS1}}$	Y3	SPDIF_TXCLK	—	GPIO7
$\overline{\text{LPC_CS2}}$	Y1	$\overline{\text{NFC_CE_1}}$	—	GPIO0
$\overline{\text{LPC_OE}}$	Y5	—	—	—
LPC_R/ $\overline{\text{W}}$	AA1	—	—	—
MA0	AD17	—	—	—
MA1	AB16	—	—	—
MA2	AE18	—	—	—
MA3	AF20	—	—	—
MA4	AD18	—	—	—
MA5	AB17	—	—	—
MA6	AE19	—	—	—
MA7	AC18	—	—	—
MA8	AF21	—	—	—
MA9	AD19	—	—	—
MA10	AF22	—	—	—
MA11	AC19	—	—	—
MA12	AE21	—	—	—
MA13	AD20	—	—	—
MA14	AB19	—	—	—
MA15	AE22	—	—	—
MBA1	AC16	—	—	—
MBA2	AF19	—	—	—
$\overline{\text{MCAS}}$	AF24	—	—	—
MCK	AF17	—	—	—

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 3 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
$\overline{\text{MCK}}$	AF18	—	—	—
MCKE	AB20	—	—	—
$\overline{\text{MCS}}$	AD22	—	—	—
MDM0	AC6	—	—	—
MDM1	AE8	—	—	—
MDM2	AF13	—	—	GPIO24
MDM3	AF16	—	—	GPIO25
MDQ0	C18	—	—	—
MDQ1	AB6	—	—	—
MDQ2	AE4	—	—	—
MDQ3	AF6	—	—	—
MDQ4	AF7	—	—	—
MDQ5	AB8	—	—	—
MDQ6	AD6	—	—	—
MDQ7	AE6	—	—	—
MDQ8	AC7	—	—	—
MDQ9	AC7	—	—	—
MDQ10	AB9	—	—	—
MDQ11	AD7	—	—	—
MDQ12	AE9	—	—	—
MDQ13	AF10	—	—	—
MDQ14	AC9	—	—	—
MDQ15	AF11	—	—	—
MDQ16	AD10	—	—	GPIO0
MDQ17	AF12	—	—	GPIO1
MDQ18	AD11	—	—	GPIO2
MDQ19	AB12	—	—	GPIO3
MDQ20	AD12	—	—	GPIO4
MDQ21	AB13	—	—	GPIO5
MDQ22	AF14	—	—	GPIO6
MDQ23	AD13	—	—	GPIO7
MDQ24	AE13	—	—	GPIO16
MDQ25	AC13	—	—	GPIO17
MDQ26	AF15	—	—	GPIO18
MDQ27	AB14	—	—	GPIO19
MDQ28	AE16	—	—	GPIO20

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 4 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
MDQ29	AD15	—	—	GPIO21
MDQ30	AC15	—	—	GPIO22
MDQ31	AB15	—	—	GPIO23
MDQS0	AD5	—	—	—
MDQS1	AD8	—	—	—
MDQS2	AC11	—	—	GPIO26
MDQS3	AD14	—	—	GPIO27
MODT	AC21	—	—	—
$\overline{\text{MRAS}}$	AF23	—	—	—
MVREF	AB11	—	—	—
MVTT0	AB7	—	—	—
MVTT1	AF9	—	—	—
MVTT2	AE11	—	—	—
MVTT3	AE14	—	—	—
$\overline{\text{MWE}}$	AD21	—	—	—
NFC_ALE	H4	SDHC_D0	LPC_AX07	GPIO20
$\overline{\text{NFC_CE_0}}$	H3	$\overline{\text{LPC_CS3}}$	PSC_MCLK_IN	GPIO26
NFC_CLE	H5	SDHC_D1_IRQ	LPC_AX06	GPIO21
NFC_R/ $\overline{\text{B}}$	H1	SDHC_CMD	LPC_AX08	GPIO19
$\overline{\text{NFC_RE}}$	G2	SDHC_D3_CD	LPC_AX04	GPIO23
$\overline{\text{NFC_WE}}$	G3	SDHC_D2	LPC_AX05	GPIO22
NFC_WP	G4	SDHC_CLK	LPC_AX09	GPIO18
$\overline{\text{PATA_CE1}}$	K1	$\overline{\text{LPC_CS4}}$	—	GPIO9
$\overline{\text{PATA_CE2}}$	L5	$\overline{\text{LPC_CS5}}$	—	GPIO10
$\overline{\text{PATA_DACK}}$	H2	SDHC_D3_CD	LPC_AX04	GPIO17
PATA_DRQ	J4	SDHC_D2	LPC_AX05	GPIO16
PATA_INTRQ	J3	SDHC_D1_IRQ	LPC_AX06	GPIO15
PATA_IOCHRDY	J2	SDHC_D0	LPC_AX07	GPIO14
$\overline{\text{PATA_IOR}}$	J1	SDHC_CLK	—	GPIO12
$\overline{\text{PATA_IOW}}$	K5	SDHC_CMD	LPC_AX08	GPIO13
PATA_ISOLATE	K3	CAN3_TX	—	GPIO11
PCI_AD00	U24	—	FEC_COL	GPIO3
PCI_AD01	V26	VIU_DATA9	FEC_MDIO	GPIO2
PCI_AD02	U25	VIU_DATA8	FEC_MDC	GPIO1
PCI_AD03	R22	VIU_DATA6	FEC_CRS	GPIO0
PCI_AD04	U26	VIU_PIX_CLK	FEC_TX_ER	GPIO27

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 5 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
PCI_AD05	T24	—	FEC_TX_EN	GPIO26
PCI_AD06	R23	—	FEC_RX_DV	GPIO25
PCI_AD07	T26	VIU_DATA7	FEC_RX_ER	GPIO24
PCI_AD08	R26	—	FEC_TX_CLK	GPIO23
PCI_AD09	P23	—	FEC_RX_CLK	GPIO22
PCI_AD10	R24	—	FEC_RXD_0	GPIO21
PCI_AD11	R25	VIU_DATA5	FEC_RXD_1	GPIO20
PCI_AD12	P26	VIU_DATA4	FEC_RXD_2	GPIO19
PCI_AD13	P24	VIU_DATA3	FEC_RXD_3	GPIO18
PCI_AD14	P25	—	FEC_TXD_0	GPIO17
PCI_AD15	N26	VIU_DATA2	FEC_TXD_1	GPIO16
PCI_AD16	L22	VIU_DATA1	FEC_TXD_2	GPIO15
PCI_AD17	K25	VIU_DATA0	FEC_TXD_3	GPIO14
PCI_AD18	J26	—	USB1_DIR	GPIO13
PCI_AD19	K24	—	USB1_CLK	GPIO12
PCI_AD20	J25	—	USB1_NEXT	GPIO11
PCI_AD21	H26	—	USB1_STOP	GPIO10
PCI_AD22	K23	—	USB1_DATA0	GPIO9
PCI_AD23	J24	—	USB1_DATA1	GPIO8
PCI_AD24	H24	—	USB1_DATA2	GPIO7
PCI_AD25	J23	—	USB1_DATA3	GPIO6
PCI_AD26	G25	—	USB1_DATA4	GPIO5
PCI_AD27	J22	—	USB1_DATA5	GPIO4
PCI_AD28	F26	—	USB1_DATA6	GPIO3
PCI_AD29	G24	—	USB1_DATA7	GPIO2
PCI_AD30	F24	USB0_DIR	DIU_LD23	GPIO1
PCI_AD31	H22	USB0_CLK	DIU_LD22	GPIO0
PCI_C/ $\overline{\text{BE}}$ 0	P22	USB0_DATA5	DIU_LD03	GPIO4
PCI_C/ $\overline{\text{BE}}$ 1	N24	USB0_DATA4	DIU_LD02	GPIO5
PCI_C/ $\overline{\text{BE}}$ 2	L24	USB0_DATA3	DIU_LD00	GPIO6
PCI_C/ $\overline{\text{BE}}$ 3	G26	USB0_DATA2	DIU_LD01	GPIO7
PCI_CLK	C26	—	DIU_LD14	GPIO15
$\overline{\text{PCI_DEVSEL}}$	L26	—	DIU_LD06	GPIO19
$\overline{\text{PCI_FRAME}}$	M23	—	DIU_LD10	GPIO17
$\overline{\text{PCI_GNT0}}$	E25	—	DIU_LD12	GPIO12
$\overline{\text{PCI_GNT1}}$	G22	—	DIU_LD19	GPIO10

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 6 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
PCI_GNT2	E24	—	DIU_LD21	GPIO8
PCI_IDSEL	K22	—	DIU_LD07	GPIO18
PCI_INTA	U23	—	DIU_LD15	GPIO14
PCI_IRDY	K26	USB0_DATA7	DIU_LD05	GPIO20
PCI_PAR	N22	USB0_DATA0	DIU_LD09	GPIO23
PCI_PERR	M25	USB0_STOP	DIU_LD16	GPIO24
PCI_REQ0	G23	—	DIU_LD11	GPIO13
PCI_REQ1	E26	—	DIU_LD18	GPIO11
PCI_REQ2	D26	—	DIU_LD20	GPIO9
PCI_RST_OUT	F22	—	DIU_LD13	GPIO16
PCI_SERR	M26	USB0_NEXT	DIU_LD17	GPIO25
PCI_STOP	M24	USB0_DATA1	DIU_LD08	GPIO22
PCI_TRDY	M22	USB0_DATA6	DIU_LD04	GPIO21
PORESET	W23	—	—	—
PSC0_0	D16	FEC_COL	USB0_DATA7	GPIO8
PSC0_1	A17	FEC_TX_EN	USB0_DATA6	GPIO9
PSC0_2	E15	FEC_TX_CLK	USB0_DATA5	GPIO10
PSC0_3	C16	FEC_TXD_0	USB0_DATA4	GPIO11
PSC0_4	B16	FEC_TXD_1	USB0_DATA3	GPIO0/GPT0
PSC10_0	C13	—	DIU_LD11	GPIO8
PSC10_1	B13	—	DIU_LD12	GPIO9
PSC10_2	A13	—	DIU_LD13	GPIO10
PSC10_3	C12	—	DIU_LD14	GPIO11
PSC10_4	E12	—	DIU_LD15	GPIO2/GPT2
PSC11_0	A12	—	DIU_LD18	GPIO12
PSC11_1	B11	—	DIU_LD19	GPIO13
PSC11_2	C11	—	DIU_LD20	GPIO14
PSC11_3	E11	—	DIU_LD21	GPIO15
PSC11_4	D11	—	DIU_DE	GPIO3/GPT3
PSC1_0	C15	FEC_TXD_2	USB0_DATA2	GPIO12
PSC1_1	A16	FEC_TXD_3	USB0_DATA1	GPIO13
PSC1_2	E14	FEC_MDC	USB0_DATA0	GPIO14
PSC1_3	A15	FEC_RX_ER	USB0_STOP	GPIO15
PSC1_4	D14	FEC_RXD_3	USB0_NEXT	GPIO1/GPT1
PSC2_0	C14	FEC_RXD_2	USB0_CLK	GPIO16
PSC2_1	B14	FEC_RXD_1	USB0_DIR	GPIO17

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 7 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
PSC2_2	E13	FEC_RXD_0	—	GPIO18
PSC2_3	A14	FEC_MDIO	—	GPIO19
PSC2_4	D13	FEC_RX_CLK	—	GPIO2/GPT2
PSC3_0	AF3	USB1_DATA0	—	GPIO20
PSC3_1	AB5	USB1_DATA1	—	GPIO21
PSC3_2	AC4	USB1_DATA2	—	GPIO22
PSC3_3	AD4	USB1_DATA3	—	GPIO23
PSC3_4	AF4	$\overline{\text{LPC_CS6}}$	VIU_PIX_CLK	GPIO3/GPT3
PSC4_0	AB1	USB1_DATA4	VIU_DATA0	GPIO24
PSC4_1	AA3	USB1_DATA5	VIU_DATA1	GPIO25
PSC4_2	AB3	USB1_DATA6	VIU_DATA2	GPIO26
PSC4_3	AA5	USB1_DATA7	VIU_DATA3	GPIO27
PSC4_4	AC2	$\overline{\text{NFC_CE2}}$	VIU_DATA4	GPIO4/GPT4
PSC5_0	AC1	USB1_CLK	VIU_DATA5	GPIO8
PSC5_1	AC3	USB1_NEXT	VIU_DATA6	GPIO9
PSC5_2	AD1	USB1_STOP	VIU_DATA7	GPIO10
PSC5_3	AD2	USB1_DIR	VIU_DATA8	GPIO11
PSC5_4	AE3	$\overline{\text{NFC_CE3}}$	VIU_DATA9	GPIO5/GPT5
PSC6_0	A11	LPC_TSI21	DIU_CLK	GPIO12
PSC6_1	C10	LPC_TSI22	DIU_HSYNC	GPIO13
PSC6_2	A10	—	—	GPIO14
PSC6_3	B9	—	—	GPIO15
PSC6_4	A9	LPC_TS	DIU_VSYNC	GPIO6/GPT6
PSC7_0	B8	SDHC_CMD	DIU_LD23	GPIO16
PSC7_1	E10	SDHC_D0	DIU_LD22	GPIO17
PSC7_2	C8	SDHC_D1_IRQ	DIU_LD17	GPIO18
PSC7_3	A8	SDHC_D2	DIU_LD16	GPIO19
PSC7_4	A7	SDHC_D3_CD	DIU_LD09	GPIO7/GPT7
PSC8_0	E9	—	DIU_LD08	GPIO20
PSC8_1	D8	—	DIU_LD01	GPIO21
PSC8_2	C7	—	DIU_LD00	GPIO22
PSC8_3	B6	—	DIU_LD02	GPIO23
PSC8_4	E8	SDHC_CLK	DIU_LD03	GPIO0/GPT0
PSC9_0	C6	—	DIU_LD04	GPIO24
PSC9_1	D7	—	DIU_LD05	GPIO25
PSC9_2	E7	—	DIU_LD06	GPIO26

Table 3-2. MPC5121e Alphabetical Signal Reference (Sheet 8 of 8)

Pin Name Function 1	Package Pin Number	Function 2	Function 3	Function 4
PSC9_3	D6	—	DIU_LD07	GPIO27
PSC9_4	K24	—	DIU_LD10	GPIO1/GPT1
PSC_MCLK_IN	C17	—	DIU_DE	GPIO6/GPT6
RTC_XTAL1	C20	—	—	—
RTC_XTALO	A20	—	—	—
SATA_ANAVIZ	E5	—	—	—
SATA_RESREF	E4	—	—	—
SATA_RXN	A4	—	—	—
SATA_RXP	A5	—	—	—
SATA_TXN	E1	—	—	—
SATA_TXP	F1	—	—	—
SATA_XTALI	C3	—	—	—
SATA_XTALO	C2	—	—	—
SPDIF_RX	AC24	FEC_CRS	DIU_HSYNC	GPIO0
SPDIF_TX	AD24	FEC_TX_ER	DIU_VSYNC	GPIO27
SPDIF_TXCLK	AB21	FEC_RX_DV	DIU_CLK	GPIO26
SRESET	V22	—	—	—
SYS_XTALI	V24	—	—	—
SYS_XTALO	W26	—	—	—
TCK	AB26	—	—	—
TDI	Y23	—	—	—
TDO	W22	—	—	—
TEST	W25	—	—	—
TMS	Y25	—	—	—
TRST	AA26	—	—	—
USB_PHY_DM	A22	—	—	—
USB_PHY_DP	A23	—	—	—
USB_PHY_DRVVBUS	A21	—	—	—
USB_PHY_RREF	E22	—	—	—
USB_PHY_TPA	A24	—	—	—
USB_PHY_UID	E19	—	—	—
USB_PHY_VBUS	—	—	—	—
USB_PHY_XTALI	C24	—	—	—
USB_PHY_XTALO	B24	—	—	—
USB_PHY_VBUS_PWR_FAULT	B21	—	—	—

3.2 Output Signal States During Reset

When a system reset is recognized ($\overline{\text{PORESET}}$ or $\overline{\text{HRESET}}$ are asserted), the MPC5121e aborts all current internal and external transactions, and it releases all bidirectional I/O signals to a high-impedance state. See [Chapter 4, “Reset,”](#) for a complete description of the reset functionality.

During reset, the MPC5121e ignores most input signals (except for reset configuration signals [Table 3-3](#)) and drives most of the output-only signals to an inactive state. [Table 3-3](#) shows states of the output-only signals.

Table 3-3. Output Signal States During System Reset

Interface	Signal	State During Reset
MDM[3:0]	DDR data mask	high-Z
MDQ[31:0]	DDR data	high-Z
MDQS[3:0]	DDR data strobe	high-Z
MBA[2:0]	DDR bank select	All 0
MA[15:0]	DDR address	All 0
$\overline{\text{MWE}}$	DDR write enable	1
$\overline{\text{MRAS}}$	DDR row address strobe	1
$\overline{\text{MCAS}}$	DDR column address strobe	1
$\overline{\text{MCS}}$	DDR chip select	1
MCKE	DDR clock enable	0
MCK	DDR differential clock	0
$\overline{\text{MCK}}$	DDR differential clock	0
MODT	DRAM On-Die Termination	0
$\overline{\text{LPC_OE}}$	LocalPlus output enable	1
$\overline{\text{LPC_R/W}}$	LocalPlus read/write bar	1
EMB_AX[0]	External Memory Bus address extension 0/LocalPlus Address latch/Parallel ATA address 0	0
TDO	Test data out	high-Z
$\overline{\text{CKSTP_OUT}}$	Check Stop output	1
$\overline{\text{PCI_RST}}$	PCI reset output	0
$\overline{\text{PCI_INTA}}$	PCI interrupt output	Z
$\overline{\text{LPC_ACK}}$	LPC Acknowledge	Pullup resistor
$\overline{\text{LPC_CS0}}$	LPC Chip Select 0	Pullup resistor
$\overline{\text{LPC_CS1}}$	LPC Chip Select 1	Pullup resistor
$\overline{\text{LPC_CS2}}$	LPC Chip Select 2	Pullup resistor
$\overline{\text{NFC_CE0}}$	NFC Chip Enable 0	Pullup resistor
$\overline{\text{PATA_CE1}}$	PATA Chip Select 1	Pullup resistor
$\overline{\text{PATA_CE2}}$	PATA Chip Select 2	Pullup resistor
$\overline{\text{PATA_ISOLATE}}$	PATA Isolation	Pullup resistor
J1850_RX	J1850 Receive port	Pullup resistor

The control device's signal multiplexing is documented in [Chapter 22, “IO Control”](#).

Chapter 4

Reset

4.1 Introduction

The MPC5121e has two reset flows, SRESET and HRESET. These flows can be initiated by the following events:

- Power on reset ($\overline{\text{PORESET}}$)
- Hardware reset ($\overline{\text{HRESET}}$)
- Soft reset ($\overline{\text{SRESET}}$)
- JTAG initiated reset (power on reset, hardware reset, and soft reset)¹
- Checkstop (MCP) event
- Watchdog timer (WDT) module
- Bus monitor
- Software write to [Section 4.7.6, “Reset Control Register \(RCR\)”](#).²

4.2 HRESET Flow

HRESET provides a mechanism to initialize all clocks and peripherals to the initial values. This flow does not sample the reset configuration word.

4.2.1 Sources

The following sources may initiate an HRESET sequence:

- $\overline{\text{PORESET}}$ input signal
- $\overline{\text{HRESET}}$ input signal
- Watchdog timer (WDT) module
- Bus monitor
- Checkstop event³
- Software write to the RESET module⁴

1. The JTAG initiated reset (power on reset, hardware reset, and soft reset) is independent of the reset state of the JTAG controller ($\overline{\text{TRST}}$).

2. The reset control register in the RESET module may initiate a SRESET or HRESET sequence.

3. Checkstop may be initiated when the e300 core enters the checkstop state. This state may be masked at in the RESET module, e300 core, or IPIC.

4. The reset configuration register in the RESET module may initiate a SRESET or HRESET sequence.

4.2.2 Impacts

When a HRESET sequence is initiated, the following occurs:

- $\overline{\text{HRESET}}$ pin is asserted by the device
- $\overline{\text{SRESET}}$ pin is asserted by the device.
- MSR[IP] bit in the e300 core is updated to reflect vector table location
- PLLs reloads programming information requiring the PLL re-lock to the reference clock signal
- Clock dividers are initialized
- Memory map initializes to reset state
- All peripheral logic asserts reset unless otherwise noted¹
- Reset source is captured in the RESET module
- The e300 core resets fetching instructions from the vector indicated by the RST_CONF word
- Real time clock shadow registers and time, date, alarm, and stopwatch registers initialize to reset state. Other RTC registers are not reset.

4.3 SRESET Flow

SRESET provides a mechanism to shorten the boot flow by bypassing initialization of boot peripherals and clocks (see [Table 4-1](#)). Timing for SRESET can be found in the data sheet.

4.3.1 Sources

The following sources initiate an SRESET sequence:

- $\overline{\text{SRESET}}$ input signal
- JTAG JSRS command
- Software write to the RESET module

4.3.2 Impacts

When SRESET sequence is initiated, the following occurs:

- $\overline{\text{SRESET}}$ pin is asserted by the device
- The following peripherals are not affected by reset:
 - IO control and pin multiplexing
 - Clock control registers
 - Memory access windows (XLBMEN)
 - Local plus memory controller (LPC)
 - DRAM controller (MDDRC)
 - RTC registers on the V_{BAT} power domain (see [Section 4.6.2, “RTC at Reset”](#))

1. The RTC timer mechanism retains state as long as V_{bat} provides power to the RTC module

- Reset source is captured in the RESET module
- MSR[IP] bit in the e300 core is not updated
- The e300 core start execution at the reset vector
- See e300 core manual for core impact of SRESET

4.4 (PORESET) Power-On Initialization

During the power-up sequence the $\overline{\text{PORESET}}$ pin must be asserted by an external device for a minimum of 32 XTAL clock cycles. The oscillator input (XTALI) must be stable prior to $\overline{\text{PORESET}}$ de-assertion. After PORESET has been qualified (see [Section 4.5, “Reset of Internal Peripherals”](#)), the HRESET flow is started.

4.5 Reset of Internal Peripherals

[Table 4-1](#) summarizes each peripheral and the relation to each reset flow.

Table 4-1. Peripheral Versus Reset

Peripheral	$\overline{\text{HRESET}}^1$	$\overline{\text{SRESET}}$
AXE	—	•
BDLC	—	•
CFM	—	•
CLOCK	•	—
CSBARB	—	•
DIU	—	•
DMA	—	•
e300 ²	•	•
EMB	—	•
FEC	—	•
FIFOC	—	•
FUSE	•	•
GPIO	•	—
GPT ³	•	•
I2C	—	•
IIM	•	—
IO_CONTROL	•	—
IPIC	—	•
LPC	•	—
MBX	—	•
MDDRC	•	—
MEM	—	•
MEMMAP	•	—

Table 4-1. Peripheral Versus Reset (continued)

Peripheral	$\overline{\text{HRESET}}^1$	$\overline{\text{SRESET}}$
MSCAN	—	•
NFC	—	•
PATA	—	•
PCI ⁴	•	•
PCI_DMA	•	•
PCI_IOS	•	•
PMC	—	•
PSC	—	•
RTC ⁵	—	•
RESET	•	—
SAP	—	•
SATA	—	•
SATA_PHY ⁶	—	—
SDHC	—	•
SPDIF	—	•
TEMPSENS	—	—
TLM ⁷	—	—
TPM	—	•
USB	—	•
USB_PHY ⁸	—	—
WDT	—	•

¹ $\overline{\text{PORESET}}$ causes the same effect as $\overline{\text{HRESET}}$.

² The e300 core supports two reset configurations. See the e300 core user's manual for details.

³ GPT supports two different reset configurations for each reset type.

⁴ PCI and submodules are effected by $\overline{\text{SRESET}}$ and $\overline{\text{HRESET}}$.

⁵ RTC is reset by V_{BAT} Power-On-Reset

⁶ SATA_PHY is reset by V_{SATA} Power-On-Reset

⁷ TLM can only be reset through use of the JTAG ($\overline{\text{TRST}}$). See [Section 4.6.3, "JTAG Reset"](#).

⁸ USB_PHY is reset by the USB controller

4.6 Reset Configuration Word (RST_CONF)

The RST_CONF word is latched when the PORESET becomes qualified. This controls the boot configuration of the device. Each RST_CONF pin MUST have external pull-up/pull-down devices which ensure that the device enters the desired mode of operation. The value latched into the device at reset may be verified by access to the RCWLR & RCWHR registers (see [Section 4.7, "Memory Map"](#)).

Available modes include:

- PCI arbiter

- Test modes
- Boot interface selection
- MUX flash mode
- NOR flash port size
- NAND flash page size
- NAND flash port size
- Core PLL programming
- System PLL programming
- Clock divider

Table 4-2. Reset Configuration Word

Reset Parameter	Signal	Description
RST_CONF_BMS	EMB_AD[5]	Boot mode select — Selects e300 boot vector and configures default value for LPC CS0 or NFC base address. See Section 4.6.1, “BMS Operation”
RST_CONF_ROMLOC	EMB_AD[1:0]	Selects boot device ¹ 00 LPC boot 01 NAND (NFC) boot 10 reserved 11 NAND (NFC) boot see also definition of RST_CONF_NFC_PS
RST_CONF_SWEN	EMB_AD[2]	Enables Watchdog Timer at reset 0 Disabled 1 Enabled
RST_CONF_LPC_MX	EMB_AD[16]	LPC Mux mode configuration 0 Non-multiplexed mode 1 Multiplexed mode
RST_CONF_LPC_AX	EMB_AD[9:8]	LPC address extension mode 00 No LPC Address Extension 01 Use LPC_AX[pata] 10 Use LPC_AX[nfc] 11 Reserved
RST_CONF_LPC_DBW	EMB_AD[19:18]	LPC Data Port Size 00 8 bit 10 Reserved 01 16 bit 11 32 bit
RST_CONF_NFC_PS	EMB_AD[20]	NAND Flash Page Size if RST_CONF_ROMLOC = 01 then RST_CONF_NFC_PS defines the page size: 0 512 bytes page size 1 2 Kbytes page size if RST_CONF_ROMLOC = 11 then RST_CONF_NFC_PS defines the spare size with a fixed page size of 4K: 0 64 bytes spare size 1 218 bytes spare size

Table 4-2. Reset Configuration Word (continued)

Reset Parameter	Signal	Description
RST_CONF_NFC_DBW	EMB_AD[21]	NFC Data Port Size 0 8 bit 1 16 bit
RST_CONF_CKS_IN	EMB_AD[22]	Checkstop 0 Checkstop input disabled 1 Checkstop input enabled
RST_CONF_COREPLL	EMB_AD[13:10]	Core PLL Multiply factor See clock module for programming options
RST_CONF_SYSPLL	EMB_AD[26:23]	System PLL Multiply factor See clock module for programming options
RST_CONF_SYSOSCEN	EMB_AX02	Oscillator Bypass Mode 0 System Oscillator bypass mode 1 System Oscillator mode
RST_CONF_SYSDIV	LPC_AX[3], EMB_AD[31:27]	System PLL divider See clock module for programming options
RST_CONF_TLE	EMB_AD[6]	Endian Mode 0 Big Endian Mode 1 Little Endian Mode
RST_CONF_LPC_WA	EMB_AD[17]	LPC word/byte addressing 0 Byte addressing 1 Word addressing
RST_CONF_PCI66EN	EMB_AD[7]	Enable 66 MHz PCI Operation ² 0 PCI 33 MHz 1 PCI 66 MHz
RST_CONF_EMB_AD14	EMB_AD[14]	Reserved — must be connected to 1
RST_CONF_PCIARB	EMB_AD[15]	Internal PCI Arbiter signals 0 Disabled 1 Enabled
RST_CONF_TPR	EMB_AD[3]	Factory Test Mode 0 Disabled (normal operation) 1 Enabled (Freescale factory test only)
RST_CONF_COREDIS	EMB_AD[4]	Core Disable Mode ³ 0 Disabled (normal operation) 1 Enabled (Freescale factory test only)

¹ LPC boot together with 4-Kbyte page NAND flash devices is not supported

² Provides the required IO timing for the different PCI modes

³ This mode is provided for Freescale factory testing only

4.6.1 BMS Operation

The boot mode select bit determines the default value of LPC CS0 and provide the reset vector to the e300 core. The e300 MSR[IP] bit reflects the state which is latched by the BMS bit. The BMS bit indicates to the e300 where in memory to fetch the first instruction.

Table 4-3. BMS Impact on Boot Vector

Parameter	BMS = 0 (boot low)	BMS = 1 (boot high)
e300 Boot Vector	0x00000100	0xFFFF00100

Table 4-4. BMS Impact On Memory Windows

Parameter	BMS = 0 (boot low)	BMS = 1 (boot high)
LPC CSBOOT Start	0x00000000	0xFF800000
LPC CSBOOT End	0x0007FFFF	0xFFFFFFFF
NFC Base Address	0x00000000	0xFFFF0000

4.6.2 RTC at Reset

The RTC module contains registers which are located on the V_{BAT} power domain which are not effected by system level reset functions. These registers can only be reset by removing power from V_{BAT} .

4.6.3 JTAG Reset

The JTAG state machine is reset independently of $\overline{PORESET}$, \overline{HRESET} and \overline{SRESET} . If the JTAG connection is not used, care should be taken to ensure that \overline{TRST} is connected to a pull-up device. Even if your system does not utilize a JTAG connector the JTAG connection must be tied to a defined state or undefined behavior may occur.

4.6.4 Boot Vector Selection

The e300 boot vector may be configured through use of the RST_CONF_ROM_LOC and RST_CONF_BMS pins at reset (see [Section 4.6, “Reset Configuration Word \(RST_CONF\)”](#)). These pins allow selection of the boot memory interface and/or e300 boot vector.

4.6.5 Boot Memory Interface Selection

The e300 boot memory interface may be selected by configuring the RST_CONF_ROM_LOC pin at reset. this allows selection of either the local plus controller (LPC) or NAND flash controller as the boot memory device. See [Section 4.6.1, “BMS Operation”](#) for details. Each interface requires a unique boot strap sequence for initializing the MPC5121e. These initialization suddenness are described in the following sections.

4.6.6 LPC Initialization Sequence

This interface is utilized when the ROM_LOC configuration in the RST_CONF selects the Local Plus Interface as the boot vector. The following boot sequence works with both boot high or boot low vectors.

Table 4-5. LPC Initialization Sequence

Step		Software Region	Note
1	Configure IMMR ¹	Reset Vector (Flash)	—
2	Configure LPC Clock Dividers	Reset Vector (Flash)	—
3	Configure CS0 Access Window	Reset Vector (Flash)	—
4	Configure CS0 Timing parameters	Reset Vector (Flash)	—
5	Perform an absolute jump to the initialization routine	Reset Vector (Flash)	Relative branching should not be used
6	Initialize Memory Map	Startup (Flash)	Initialize all memory access windows and LPC chip selects
7	Initialize e300 Core	Startup (Flash)	Initialize core settings including cache policies and instruction burst capabilities. Flash should be initialized as both I/D cached in copyback mode ² .
8	Initialize system clocks	Startup (Flash)	—
9	Initialize DRAM	Startup (Flash)	—
10	Initialize IO pin muxing	Startup (Flash)	—
11	Code re-location	Startup (Flash)	Relocate code into DRAM for faster execution
12	c-runtime initialization	Startup (Flash)	This allows remaining routines
13	peripheral initialization	Startup (Flash)	Perform any initialization required by monitor or RTOS here.
14	Boot Into Application Space	DRAM	Start Application Environment (RTOS)

¹ Care must be taken to ensure that the IMMR address region does not overlap any active access windows at any time during the boot process.

² This must be changed to cache inhibited prior to entry into user code space.

4.6.7 NFC Initialization Sequence

This interface is utilized when the ROM_LOC configuration in the RST_CONF selects the NFC (NAND flash) Interface as the boot device. Care must be taken to ensure that the reset vector and NFC bootstrap fits into a single NAND flash page of 512 B or 2 Kbytes.

Table 4-6. NFC Initialization Sequence

Step		Software Region	Note
1	Configure IMMR ¹	Reset Vector (flash)	—
2	Configure DRAM & NFC Clock Dividers	Reset Vector (flash)	—
3	Configure NFC parameters	Reset Vector (flash)	—
4	Initialize DRAM	Reset Vector (flash)	Initialization should include DRAM access window as well as timings and initialization

Table 4-6. NFC Initialization Sequence (continued)

Step		Software Region	Note
5	Copy NFC bootstrap to DRAM	Reset Vector (flash)	—
6	Perform absolute jump to NFC bootstrap	NFC Bootstrap (DRAM)	Relative branch should not be used.
7	Initialize Memory Map	Startup (DRAM)	Initialize all memory access windows and LPC chip selects
8	Code re-location	Startup (DRAM)	Relocate code into DRAM for faster execution. Must use NAND flash mini-driver to access NAND flash
9	Initialize e300 Core	Startup (DRAM)	Initialize core settings including cache policies and instruction burst capabilities. Flash should be initialized as both I/D cached in copyback mode ² .
10	Initialize system clocks	Startup (DRAM)	—
11	Initialize IO pin muxing	Startup (DRAM)	—
12	c-runtime initialization	Startup (DRAM)	This allows remaining routines
13	peripheral initialization	Startup (DRAM)	Perform any initialization required by monitor or RTOS here.
14	Boot Into Application Space	(DRAM)	Start Application Environment (RTOS)

¹ Care must be taken to ensure that the IMMR address region does not overlap any active access windows at any time during the boot process.

² This must be changed to cache inhibited prior to entry into user code space.

4.7 Memory Map

The reset configuration and status registers are shown in [Table 4-7](#).

Table 4-7. Reset Configuration Registers Memory Map

Address (Offset)	Use	Access
0x00	Reset Configuration Word Low Register (RCWLR)	R
0x04	Reset Configuration Word High Register (RCWHR)	R
0x08	Reserved	—
0x0C	Reserved	—
0x10	Reset Status Register (RSR)	R/W
0x14	Reset Mode Register (RMR)	R/W
0x18	Reset Protection Register (RPR)	R/W
0x1C	Reset Control Register (RCR)	R/W
0x20	Reset Control Enable Register (RCER)	R/W
0x24–0xFC	Reserved	—

4.7.1 Reset Configuration Word Low Register (RCWLR)

The Reset Configuration Word Low Register is shown in Figure 4-1. This is a read only register that gets its values according to the reset configuration word low loaded during the reset flow.

Offset 0x00

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
					SYSPLL								COREPLL			
R					SPM F3	SPM F2	SPM F1	SPM F0					CPMF 3	CPMF 4	CPMF 5	CPMF 6
W																
Reset	0	0	0	0	EMB_ AD26	EMB_ AD25	EMB_ AD24	EMB_ AD23	0	0	0	0	EMB_ AD13	EMB_ AD12	EMB_ AD11	EMB_ AD10
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			SYSDIV													
R			SYS DIV5	SYS DIV4	SYS DIV3	SYS DIV2	SYS DIV1	SYS DIV0								
W																
Reset	0	0	LPC_ AX03	EMB_ AD31	EMB_ AD30	EMB_ AD29	EMB_ AD28	EMB_ AD27	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 4-1. Reset Configuration Word Low Register (RCWLR)

Table 4-8. RCWLR Field Descriptions

Field	Description
SPMF	System PLL multiplication factor.
CPMF	Core PLL configuration.
SYSDIV	System clock divide factor, see clock module for description.

4.7.2 Reset Configuration Word High Register (RCWHR)

The Reset Configuration Word High Register is shown in Figure 4-2. This is a read only register which get its values according to the reset configuration word high loaded during the reset flow.

Offset 0x04

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	EMB_AD14	EMB_AD07	EMB_AD15	EMB_AX02	EMB_AD04	EMB_AD05	0	0	EMB_AD02	EMB_AD01	EMB_AD00	EMB_AD03	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	EMB_AD16	EMB_AD17	EMB_AD19	EMB_AD18	EMB_AD20	EMB_AD21	EMB_AD09	EMB_AD08	EMB_AD06	EMB_AD22	0	0

= Unimplemented or Reserved

Figure 4-2. Reset Configuration Word High Register (RCWHR)

Table 4-9. RCWHR Field Descriptions

Field	Description
PCI66EN	PCI 66 enable
PCIARB	PCI internal arbiter mode
SYSOSCEN	System oscillator enable
COREDISE	Core disable mode
BMS	Boot memory space
SWEN	Software watchdog enable
ROMLOC	Boot ROM interface location
TPR	TPR pin muxing
LPC_MX	LPC muxed mode

Table 4-9. RCWHR Field Descriptions (continued)

LPC_WA	LPC word/byte address
LPC_DBW	LPC data bus width
NFC_PS	NFC page size
LPC_AX	LPC extended address bus
NFC_DBW	NFC data bus width
TLE	True little-endian
CKS_IN	Checkstop in pin muxing

NOTE

The value of fields in the reset configuration words registers (RCWLR and RCWHR) reflect only their state during the reset flow. Some of these parameters and modes may be modified by changing their values in other units' memory mapped registers. Modifying values in these other units' memory mapped registers do not affect RCWLR and RCWHR.

4.7.3 Reset Status Register (RSR)

The reset status register shown in Figure 4-5 captures various reset events in the device.

Offset 0x10												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W			SWSR	SWHR			JHRS	JSRS				CSHR	SWRS	BMRS	SRS	HRS
Reset	0	0	w1c	w1c	0	0	0	w1c	0	0	0	w1c	w1c	w1c	w1c	w1c

 = Unimplemented or Reserved

Figure 4-3. Reset Status Register (RSR)

Table 4-10. RSR Field Descriptions

Field	Description
SWSR	Software soft reset. If set, indicates that a software soft reset has occurred. cleared by writing a logic 1
SWHR	Software hard reset. If set, indicates that a software hard reset has occurred. cleared by writing a logic 1
JHRS	JTAG hard reset status. When the JTAG reset (command) request is set, JHRS is set and remains set until software clears it. JHRS is cleared by writing a logic 1 to it (writing zero has no effect). 0 No JTAG reset event occurred 1 A JTAG reset event occurred
JSRS	JTAG soft reset status. When the JTAG reset (command) request is set, JSRS is set and remains set until software clears it. JSRS is cleared by writing a logic 1 to it (writing zero has no effect). 0 No JTAG reset event occurred 1 A JTAG reset event occurred
CSHR	Check stop reset status. When the core enters a checkstop state and the checkstop reset is enabled by the RMR[CSRE], CSRS is set and it remains set until software clears it. CSRS is cleared by writing a logic 1 to it (writing zero has no effect). 0 No enabled checkstop reset event occurred 1 An enabled checkstop reset event occurred
SWRS	Software watchdog reset status. When a software watchdog expire event (which causes a reset) is detected, the SWRS bit is set and remains that way until the software clears it. SWRS is cleared by writing a logic 1. 0 No software watchdog reset event occurred 1 A software watchdog reset event has occurred
BMRS	Bus monitor reset status. When a bus monitor expire event (which causes a reset) is detected, BMRS is set and remains set until the software clears it. BMRS is cleared by writing a logic 1. 0 No bus monitor reset event has occurred 1 A bus monitor reset event has occurred
SRS	Soft reset status. When an external or internal soft reset event is detected, SRS is set and remains that way until software clears it. SRS is cleared by writing a logic 1 0 No soft reset event has occurred 1 A soft reset event has occurred Note: Soft reset induced by hard reset also sets this bit.
HRS	Hard reset status. When an external or internal hard reset event is detected, HRS is set and remains that way until software clears it. HRS is cleared by writing a logic 1 0 No hard reset event has occurred 1 A hard reset event has occurred

NOTE

The Reset Status Register accumulates reset events. This register returns to its reset value only when power-on reset occurs.

4.7.4 Reset Mode Register (RMR)

The reset mode register (RMR), shown in Figure 4-4, triggers an HRESET sequence when the e300 core enters checkstop state if the CSRE bit is set.



Figure 4-4. Reset Mode Register (RMR)

Table 4-11. RMR Field Descriptions

Field	Description
CSRE	Checkstop reset enable. The e300 core can enter checkstop mode as the result of several exception conditions. Setting CSRE configures the chip to perform a hard reset sequence when the e300 core enters checkstop state. 0 Reset not generated when core enters checkstop state. 1 Reset generated when core enters checkstop state.

4.7.5 Reset Protection Register (RPR)

The reset protection register shown in Figure 4-5, enable or disable writing to the reset control register (RCR). This register prevents unwanted resets due to unintended software writes to the reset control register (RCR). The user should write the value 0x52535445 (RSTE in ASCII) to enable. Enable indication appears in the reset control enable register (RCER[CRE]). Reading this register always returns all zeros. To disable write to the reset control register (RCR), the user should write 1 to RCER[CRE].

Offset 0x18												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RCPW															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RCPW															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-5. Reset Protection Register (RPR)

Table 4-12. RPR Field Descriptions

Field	Description
RCPW	Reset control protection word. Protects unintended software reset requests occurred because of write to the reset control register (RCR). The user should write the value 0x52535445 (RSTE in ASCII) to enable. Enable indication appears in the reset status register (RCER[CRE]). Reading this register always returns all zeros. To disable write to the reset control register (RCR), the user should write 1 to RCER[CRE].

4.7.6 Reset Control Register (RCR)

The reset control register shown in Figure 4-6, can be used by software to initiate a soft or hard reset sequence. To allow writing to this register, the user has to first enable it by writing the value 0x52535445 to the reset protection register (RPR).

Offset 0x1C												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W															SWHR	SWSR
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 4-6. Reset Control Register (RCR)

Table 4-13. RCR Field Descriptions

Field	Description
SWHR	Software hard reset. Setting this bit causes the MPC5121e to begin a hard reset flow. This bit returns to its reset state during the reset sequence, so reading it always returns 0.
SWSR	Software soft reset. Setting this bit causes the MPC5121e to begin a soft reset flow. This bit returns to its reset state during the reset sequence, so reading it always returns 0.

4.7.7 Reset Control Enable Register (RCER)

The reset control enable register shown in Figure 4-7, indicates by the CRE field that the reset protection register (RPR) was accessed with a value that enables the reset control register (RCR).

Offset 0x20 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																CRE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 4-7. Reset Control Enable Register (RCER)

Table 4-14. RCER Field Descriptions

Field	Description
CRE	Control register enabled. When set, indicates that the reset protection register (RPR) was accessed with a value that enables the reset control register (RCR). Writing 1 to this bit disables the reset control register (RCR) and clear this bit. Writing zero has no effect.

4.8 IO During Reset Assertion

When a system reset is recognized ($\overline{\text{PORESET}}$, $\overline{\text{HRESET}}$, and $\overline{\text{SRESET}}$ are asserted), the MPC5121e aborts all current internal and external transactions. All I/O signals go into a high-impedance state. Exceptions can be seen in Table 4-15.

Table 4-15. IO During Reset Assertion

Interface	Signal	State During Reset
EMB_AD[31:0]	See Section 4.6, "Reset Configuration Word (RST_CONF)"	Input
LPC_AX[3]	See Section 4.6, "Reset Configuration Word (RST_CONF)"	Input
MDM[3:0]	DDR data mask	0

Table 4-15. IO During Reset Assertion (continued)

Interface	Signal	State During Reset
MCKE	DDR clock enable	0
MCK	DDR differential clocks	0
$\overline{\text{MCK}}$	DDR differential clocks	0
MODT	DRAM on-die termination	0
$\overline{\text{LPC_OE}}$	LocalPlus output enable	1
$\text{LPC_R}/\overline{\text{W}}$	LocalPlus read/write bar	1
EMB_AX[0]	External memory bus address extension 0/LocalPlus address latch/parallel ATA address 0	0
$\overline{\text{CKSTP_OUT}}$	Check stop output	1

Chapter 5

Clocks and Low-Power Modes

5.1 Introduction

The wide range of applications supported by the MPC5121e require a complex clocking structure with different primary clock domains derived from four separate oscillator sources. Internal PLLs and clock dividers allow generation of a wide range of clock references.

Each peripheral clock may be individually controlled to minimize total power consumption of the device. Clocks may be gated individually or scaled in frequency to ensure the most efficient power profile for the user application.

5.2 System Clock Generation

The system reference is provided by a crystal oscillator that drives the system PLL. This PLL is programmed at reset by the reset configuration word (RST_CONFIG) sampled at the rising edge (deassertion) of power on reset. The SYS_PLL clock is then divided (SYS_DIV) and used as a reference to the MPC5121e cores and peripherals.

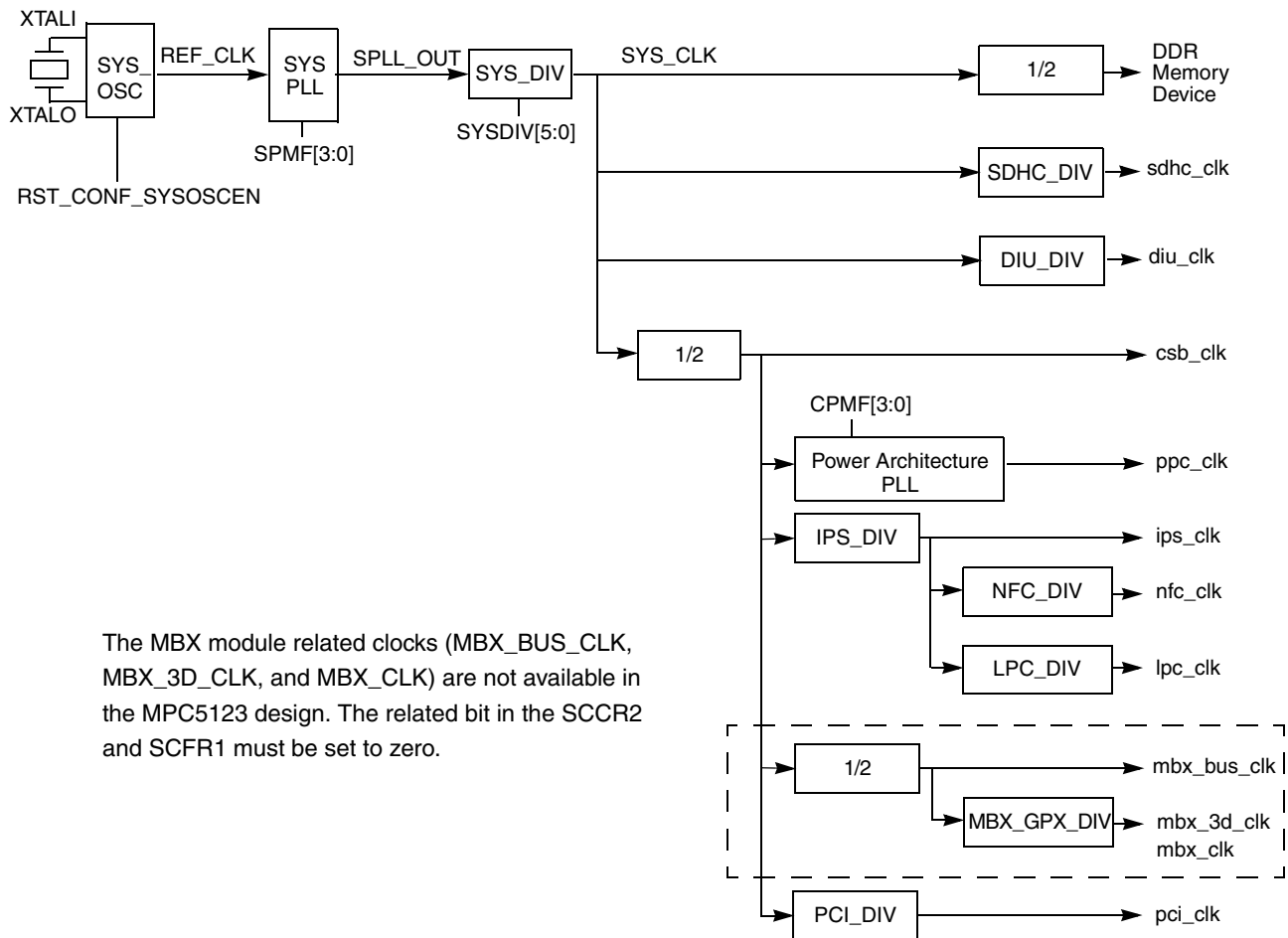


Figure 5-1. Clock System

Table 5-1. Clock Domain Programming

Domain	Programing Interface
SYS_PLL	Hardware Programmable (Section 5.2.9.1, “System PLL”)
CORE_PLL	Hardware Programmable (Section 5.2.9.2, “e300 PLL Programming Model”)
IPS_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
NFC_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
LPC_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
DIU_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
MBX_GPX_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
PCI_DIV	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)
PSC_DIV ¹	Software Programmable (Section 5.3.1, “Memory Map/Register Definition”)

¹ PSC clock generation sub-system is described in [Section 5.2.4, “PSC Clock Generation,”](#) on page 5-5.

5.2.1 Peripheral Clock Domains

The reference clock for each peripheral can be found in [Table 5-2](#).

Table 5-2. Peripheral Clock Reference

Peripheral	REFERENCE CLOCK
AXE	CSB_CLK
BDLC	IPS_CLK
CFM	CSB_CLK
CSBARB	CSB_CLK
DIU	DIU_CLK, CSB_CLK, IPS_CLK
DMA	CSB_CLK
E300	PPC_CLK
PPC_PLL	CSB_CLK
EMB	IPS_CLK, LPC_CLK, NFC_CLK
FEC	IPS_CLK
FIFOC	IPS_CLK
FUSE	IPS_CLK
GPIO	IPS_CLK
I2C	IPS_CLK
IIM	IPS_CLK
IPIC	IPS_CLK
LPC	LPC_CLK
MBX	MBX_BUS_CLK, MBX_3D_CLK, MBX_CLK
MDDRC	DDR_CLK
MEM	CSB_CLK
MSCAN	IPS_CLK ¹
NFC	NFC_CLK
PATA	IPS_CLK
PCI	PCI_CLK, CSB_CLK
PCI_DMA	CSB_CLK
PCI_IOS	CSB_CLK
PMC	IPS_CLK, REF_CLK
PRIMAN	CSB_CLK
PSC	IPS_CLK ²
RTC	IPS_CLK, RTC_CLK
SAP	IPS_CLK, TCK_CLK

Table 5-2. Peripheral Clock Reference (continued)

Peripheral	REFERENCE CLOCK
SATA	IPS_CLK, SATA_PHYCLK
SATA_PHY	SATA_OSC
SDHC	IPS_CLK, SDHC_CLK
SPDIF	IPS_CLK
SYS_PLL	REF_CLK
TLM	TCK
TPR	IPS_CLK
USB0	IPS_CLK, USB_CLK
USB1	IPS_CLK, USB_CLK
USB_PHY	USB_OSC
WDT	IPS_CLK
VIDEO_IN	CSB_CLK

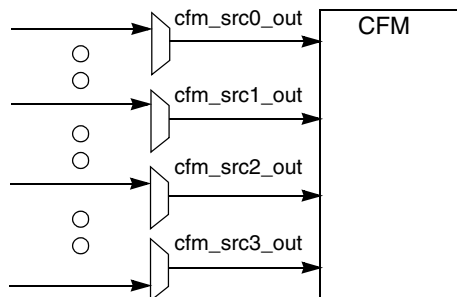
¹ For all clock sources, see [Section 5.2.5, “MSCAN Clock Generation,”](#) on page 5-5.

² For all clock sources, see [Section 5.2.4, “PSC Clock Generation,”](#) on page 5-5.

Many peripheral clocks may be disabled to reduce power consumption, see [Section 5.3.1.2, “System Clock Control Register 1 \(SCCR1\)”](#) for more information.

5.2.2 Clock Frequency Measurement (CFM) Clock Selection

The CFM module is capable of measuring the clock frequency of any PSC mclk (PSC reference clock) or PSC bit clock (PSC output clock). The CSB_CLK is then used as a reference to determine the long term frequency of the PSC clock. The CFM is capable of accepting four inputs. Each input is connected to the CFM_SRCN_OUT as indicated below.

**Figure 5-2. CFM Clock Input**

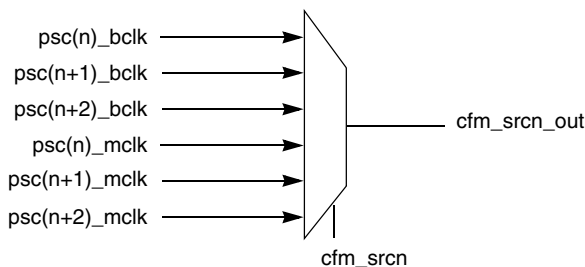


Figure 5-3. CFM Clock Input

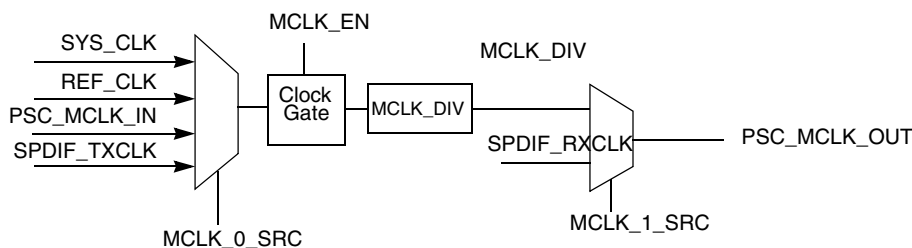
The signals indicated by psc(n) can be the MCLK reference clock or the bit clock (BCLK) output of the psc. Each mux provides selectable access to 3 PSC clock sources.

5.2.3 System Oscillator Disable

The system oscillator may be disabled so that an externally generated clock may be used as a reference. This can be performed by setting the RST_CONF_SYSOSCEN in the reset configuration word (RST_CFG) at reset.

5.2.4 PSC Clock Generation

Each PSC can select from multiple clock sources. A single clock input is provided that allows the PSC_MCLK_IN to be used as a master reference by all PSCs. Additionally, clock gating is supplied allowing the shutdown of unnecessary clocks.

Figure 5-4. PSC (MCLK) Clock Generation¹

This circuit is replicated for each PSC and can be controlled by accessing the registers starting at [Section 5.3.1.8, “PSC0 Clock Control Register \(P0CCR\)”](#).

5.2.5 MSCAN Clock Generation

Each MSCAN module can select from multiple clock sources. A single clock input is provided that allows the PSC_MCLK_IN to be used as a clock source for all MSCAN modules. Additionally, clock gating is supplied allowing the shutdown of unnecessary clocks.

1. PSC_MCLK_IN and SPDIF_TXCLK are generated by external pins. SPDIF_RXCLK is generated by the spdif module.

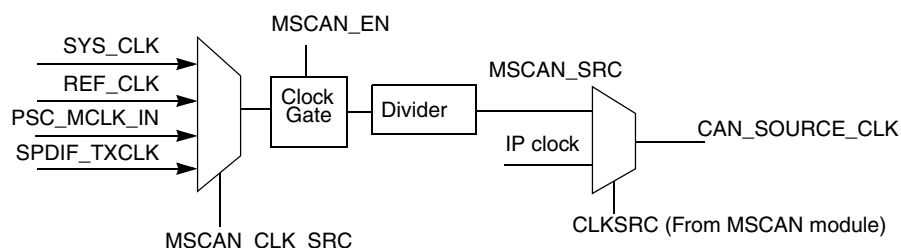


Figure 5-5. MSCAN Source Clock Generation¹

This circuit is replicated for each MSCAN module and can be controlled by accessing the registers starting at [Section 5.3.1.23, “MSCAN1Clock Control Register \(M1CCR\)”](#).

5.2.6 RTC Clock Generation

The RTC module contains circuitry on two clock and voltage domains. The V_{bat} voltage domain circuitry operates from a 32.768 KHz oscillator input. The programming interface operates from the clock reference provided by the IPBUS interface.

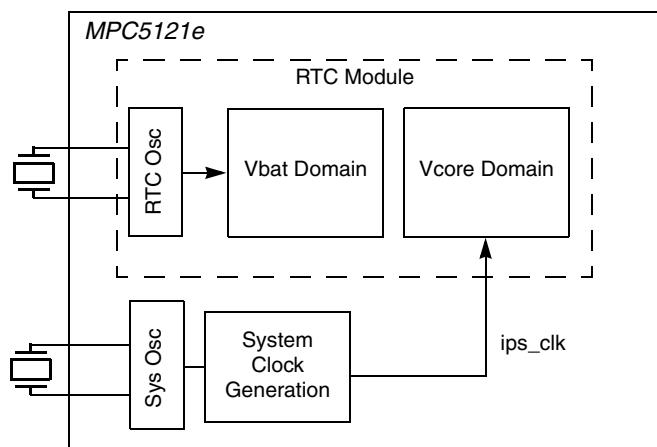


Figure 5-6. RTC Clock Generation

1. PSC_MCLK_IN and SPDIF_TXCLK are generated by external pins.

5.2.7 SATA Clock Generation

The SATA interface requires a 25 MHz crystal input that is independent of the system oscillator. The 1.5 GHz clock required by SATA physical interface (SATA PHY) is generated by this 25 MHz input.

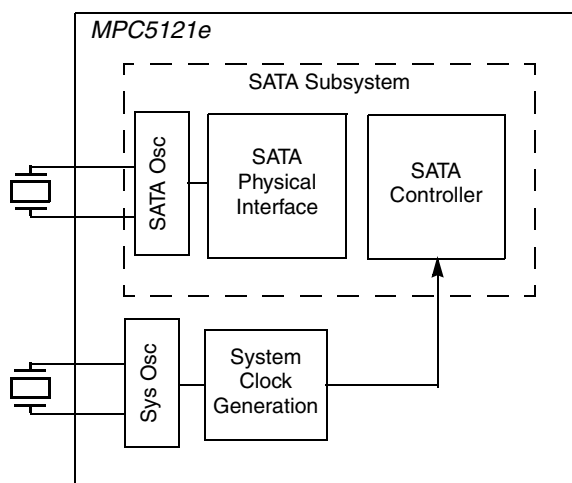


Figure 5-7. SATA Clock Generation

5.2.8 USB Clock Generation

The USB2.0 (OTG) specification requires a clock operating up to 480 MHz. This clock is derived from a dedicated oscillator input and multiplied within the USB Physical Interface for USB transmission. The USB-OTG controller operates from the `ips_clk` interface.

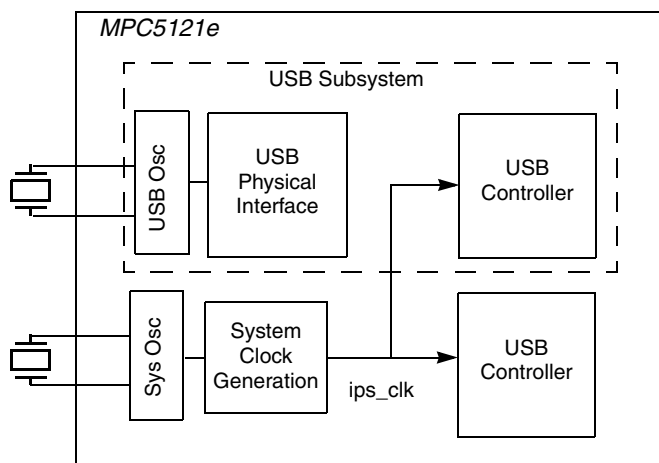


Figure 5-8. USB Clock Generation

5.2.9 System PLL and e300 PLL

5.2.9.1 System PLL

5.2.9.1.1 System PLL Multiplication Factor

The output of the system PLL conforms to the following equation (see [Figure 5-1](#)).

$$f_{\text{spll}} = \text{SPMF} * f_{\text{ref_clk}}$$

Eqn. 5-1

Table 5-3. System PLL Programming – Normal Mode

Field Name	Value (Binary)	$f_{\text{spll}}:f_{\text{ref_clk}}$
SPMF	0000	68:1
	0001	PLL BYPASS, all the clocks from CLOCK block are bypassed with PLL reference clock
	0010	12:1
	0011	16:1
	0100	20:1
	0101	24:1
	0110	28:1
	0111	32:1
	1000	36:1
	1001	40:1
	1010	44:1
	1011	48:1
	1100	52:1
	1101	56:1
	1110	60:1
	1111	64:1

5.2.9.2 e300 PLL Programming Model

Table 5-4. Core PLL Programming – Normal Mode

Field Name	Value (Binary)	f_{cpl}/f_{csb_clk}
CPMF	000?	Core PLL BYPASS/OFF
	0010	1.0:1
	0011	1.5:1
	0100	2.0:1
	2.5:1	0101
	3.0:1	0110
	3.5:1	0111
	4.0:1	1000
	4.5:1	1001
	5.0:1	1010
	5.5:1	1011
	6.0:1	1100
	6.5:1	1101
	7.0:1	1110
	7.5:1	1111

5.3 Clock Control Module

5.3.1 Memory Map/Register Definition

The clock configuration and status registers are shown in [Table 5-5](#). The access to this register is 32 bit only.

Table 5-5. Clock Configuration Registers Memory Map

Address (Offset)	Use	Access
0x00	System PLL Mode Register (SPMR)	R
0x04	System Clock Control Register 1 (SCCR1)	R/W
0x08	System Clock Control Register 2 (SCCR2)	R/W
0x0C	System Clock Frequency Register 1 (SCFR1)	R/W
0x10	System Clock Frequency Register 2 (SCFR2)	R/W
0x14	System Clock Frequency Shadow Register 2 (SCFR2S)	R/W
0x18	Bread Crumb Register (BCR)	R/W
0x1C	PSC0 Clock Control Register (P0CCR)	R/W
0x20	PSC1 Clock Control Register (P1CCR)	R/W
0x24	PSC2 Clock Control Register (P2CCR)	R/W
0x28	PSC3 Clock Control Register (P3CCR)	R/W
0x2C	PSC4 Clock Control Register (P4CCR)	R/W
0x30	PSC5 Clock Control Register (P5CCR)	R/W
0x34	PSC6 Clock Control Register (P6CCR)	R/W
0x38	PSC7 Clock Control Register (P7CCR)	R/W
0x3C	PSC8 Clock Control Register (P8CCR)	R/W
0x40	PSC9 Clock Control Register (P9CCR)	R/W
0x44	PSC10 Clock Control Register (P10CCR)	R/W
0x48	PSC11 Clock Control Register (P11CCR)	R/W
0x4C	SPDIF Clock Control Register (SPCCR)	R/W
0x50	CFM Clock Control Register (CCCR)	R/W
0x54	DIU Clock Config Register (DCCR)	R/W
0x58	MSCAN1 Clock Control Register (M1CCR)	R/W
0x5C	MSCAN2 Clock Control Register (M2CCR)	R/W
0x60	MSCAN3 Clock Control Register (M3CCR)	R/W
0x64	MSCAN4 Clock Control Register (M4CCR)	R/W
0x68 – 0xFC	Reserved.	—

5.3.1.1 System PLL Mode Register (SPMR)

Figure 5-9 shows the system PLL mode register. This is a read only register that retrieves its values from reset configuration word low loaded during the reset flow. This register is updated during a power up and PORESET sequence.

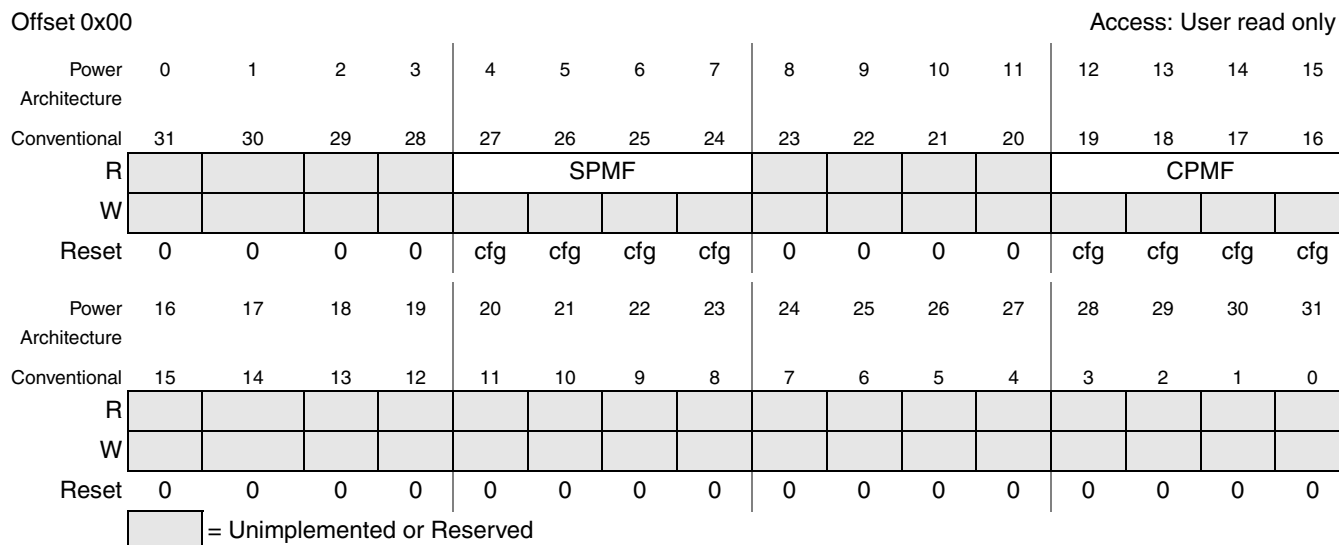


Figure 5-9. System PLL Mode Register (SPMR)

Table 5-6. SPMR Field Descriptions

Field	Description
SPMF	System PLL Multiplication Factor. See Section 5.2.9.1, “System PLL,” on page 5-8.
CPMF	Core PLL Configuration. See Section 5.2.9.2, “e300 PLL Programming Model,” on page 5-9.

5.3.1.2 System Clock Control Register 1 (SCCR1)

The system clock control register 1 shown in Figure 5-10 controls device units with a configurable clock ratio.

Offset 0x04 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CFG_EN	LPC_EN	NFC_EN	PATA_EN	PSC0_EN	PSC1_EN	PSC2_EN	PSC3_EN	PSC4_EN	PSC5_EN	PSC6_EN	PSC7_EN	PSC8_EN	PSC9_EN	PSC10_EN	PSC11_EN
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FIFOC_EN	SATA_EN	FEC_EN	TPR_EN	PCI_EN	DDR_EN										
W																
Reset	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 5-10. System Clock Control Register 1 (SCCR1)
(Register is repeated for reference.)

Table 5-7. SCCR1 Field Descriptions (Sheet 1 of 3)

Field	Description
CFG_EN	This disables access to the memory map configuration registers for IO_CONTROL and MEMMAP configuration 0 Disable 1 Enable
LPC_EN	lpc_clk Enable 0 Disable 1 Enable
NFC_EN	nfc_clk Enable 0 Disable 1 Enable
PATA_EN	pata_clk Enable 0 Disable 1 Enable
PSC0_EN	PSC0_clk Enable 0 Disable 1 Enable
PSC1_EN	PSC1 Clock Enable 0 Disable 1 Enable
PSC2_EN	PSC2 Clock Enable 0 Disable 1 Enable

Offset 0x04

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CFG_EN	LPC_EN	NFC_EN	PATA_EN	PSC0_EN	PSC1_EN	PSC2_EN	PSC3_EN	PSC4_EN	PSC5_EN	PSC6_EN	PSC7_EN	PSC8_EN	PSC9_EN	PSC10_EN	PSC11_EN
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FIFO_C_EN	SATA_EN	FEC_EN	TPR_EN	PCI_EN	DDR_EN										
W																
Reset	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 5-10. System Clock Control Register 1 (SCCR1)
 (Register is repeated for reference.)

Table 5-7. SCCR1 Field Descriptions (Sheet 2 of 3)

Field	Description
PSC3_EN	PSC3 Clock Enable 0 Disable 1 Enable
PSC4_EN	PSC4 Clock Enable 0 Disable 1 Enable
PSC5_EN	PSC5 Clock Enable 0 Disable 1 Enable
PSC6_EN	PSC6 Clock Enable 0 Disable 1 Enable
PSC7_EN	PSC7 Clock Enable 0 Disable 1 Enable
PSC8_EN	PSC8 Clock Enable 0 Disable 1 Enable
PSC9_EN	PSC9 Clock Enable 0 Disable 1 Enable
PSC10_EN	PSC10 Clock Enable 0 Disable 1 Enable
PSC11_EN	PSC11 Clock Enable 0 Disable 1 Enable

Offset 0x04

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CFG_EN	LPC_EN	NFC_EN	PATA_EN	PSC0_EN	PSC1_EN	PSC2_EN	PSC3_EN	PSC4_EN	PSC5_EN	PSC6_EN	PSC7_EN	PSC8_EN	PSC9_EN	PSC10_EN	PSC11_EN
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FIFOC_EN	SATA_EN	FEC_EN	TPR_EN	PCI_EN	DDR_EN										
W																
Reset	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 5-10. System Clock Control Register 1 (SCCR1)
(Register is repeated for reference.)

Table 5-7. SCCR1 Field Descriptions (Sheet 3 of 3)

Field	Description
FIFOC_EN	FIFOC Clock Enable, 0 Disable 1 Enable Note: If one of the PSC is used, the FIFOC clock must be enabled
SATA_EN	SATA Clock Enable, controls the SATA controller clock, not the SATA-PHY clock 0 Disable 1 Enable
FEC_EN	FEC Clock Enable 0 Disable 1 Enable
TPR_EN	TPR and SAP Clock Enable 0 Disable 1 Enable Note: Always set this bit to 1 for all normal user modes. It can be disabled if no debugger is needed.
PCI_EN	PCI Clock Enable 0 Disable 1 Enable
DDR_EN	MDDRC Clock Enable 0 Disable 1 Enable

5.3.1.3 System Clock Control Register 2 (SCCR2)

The system clock control register 2 shown in Figure 5-11 controls device units with a configurable clock ratio.

Offset 0x08

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DIU_EN	AXE_EN	MEM_EN	USB1_EN	USB2_EN	I2C_EN	BDLC_EN	SDH_C_EN	SPDI_F_EN	MBX_BUS_EN	MBX_EN	MBX_3D_EN	IIM_EN	VIDEO_in_EN		
W																
Reset	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 5-11. System Clock Control Register 2 (SCCR2)
(Register is repeated for reference.)

Table 5-8. SCCR2 Field Descriptions

Field	Description
DIU_EN	DIU Clock Enable 0 Disable 1 Enable
AXE_EN	AXE Clock Enable 0 Disable 1 Enable
MEM_EN	MEM Clock Enable 0 Disable 1 Enable
USB2	USB1 Clock Enable 0 Disable 1 Enable
USB1	USB2 Clock Enable 0 Disable 1 Enable
I2C_EN	I2C Clock Enable 0 Disable 1 Enable
BDLC_EN	BDLC & MSCAN Clock Enable 0 Disable 1 Enable

Offset 0x08

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DIU_EN	AXE_EN	MEM_EN	USB1_EN	USB2_EN	I2C_EN	BDLC_EN	SDHC_EN	SPDIF_EN	MBX_BUS_EN	MBX_EN	MBX_3D_EN	IIM_EN	VIDEO_IN_EN		
W																
Reset	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 5-11. System Clock Control Register 2 (SCCR2)
(Register is repeated for reference.)

Table 5-8. SCCR2 Field Descriptions (continued)

Field	Description
SDHC_EN	SDHC Clock Enable 0 Disable 1 Enable
SPDIF_EN	SPDIF Clock Enable 0 Disable 1 Enable
MBX_BUS_EN	MBX BUS Clock Enable 0 Disable 1 Enable
MBX_EN	MBX Clock Enable 0 Disable 1 Enable
MBX_3D_EN	MBX 3D Clock Enable 0 Disable 1 Enable
IIM_EN	IIM Clock Enable 0 Disable 1 Enable
VIDEO_IN_EN	VIDEO_IN Clock Enable 0 Disable 1 Enable

5.3.1.4 System Clock Frequency Register 1 (SCFR1)

The system clock frequency register 1 shown in Figure 5-12 controls device units with a configurable clock ratio.

Offset 0x0C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0

= Unimplemented or Reserved

Figure 5-12. System Clock Frequency Register 1 (SCFR1)

Table 5-9. SCFR1 Field Descriptions (Sheet 1 of 3)

Field	Description										
IPS_DIV	IPS Clock Divide Ratio Valid Settings <table> <tr> <th>IPS_DIV</th><th>Bit Encoding</th></tr> <tr> <td>1/2</td><td>010</td></tr> <tr> <td>1/3</td><td>011</td></tr> <tr> <td>1/4</td><td>100</td></tr> <tr> <td>1/6</td><td>110</td></tr> </table>	IPS_DIV	Bit Encoding	1/2	010	1/3	011	1/4	100	1/6	110
IPS_DIV	Bit Encoding										
1/2	010										
1/3	011										
1/4	100										
1/6	110										
PCI_DIV	PCI Clock Divide Ratio Valid Settings <table> <tr> <th>PCI_DIV</th><th>Bit Encoding</th></tr> <tr> <td>1/2</td><td>010</td></tr> <tr> <td>1/3</td><td>011</td></tr> <tr> <td>1/4</td><td>100</td></tr> <tr> <td>1/6</td><td>110</td></tr> </table>	PCI_DIV	Bit Encoding	1/2	010	1/3	011	1/4	100	1/6	110
PCI_DIV	Bit Encoding										
1/2	010										
1/3	011										
1/4	100										
1/6	110										

Offset 0x0C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							IPS_DIV		PCI_DIV							MBX_GPX_DIV
W																
Reset	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MBX_GPX_DIV		LPC_DIV			NFC_DIV			DIU_DIV							
W																
Reset	0	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0

= Unimplemented or Reserved

Figure 5-12. System Clock Frequency Register 1 (SCFR1)

Table 5-9. SCFR1 Field Descriptions (Sheet 2 of 3)

Field	Description										
MBS_GPX_DIV	MBX 3D Clock Divide Ratio <table border="1"> <thead> <tr> <th>MBS_GPX_DIV</th><th>Bit Encoding</th></tr> </thead> <tbody> <tr> <td>1/1</td><td>001</td></tr> <tr> <td>1/2</td><td>010</td></tr> <tr> <td>1/3</td><td>011</td></tr> <tr> <td>1/4</td><td>100</td></tr> </tbody> </table>	MBS_GPX_DIV	Bit Encoding	1/1	001	1/2	010	1/3	011	1/4	100
MBS_GPX_DIV	Bit Encoding										
1/1	001										
1/2	010										
1/3	011										
1/4	100										
LPC_DIV	LPC Clock Divide Ratio <table border="1"> <thead> <tr> <th>LPC_DIV</th><th>Bit Encoding</th></tr> </thead> <tbody> <tr> <td>1/1</td><td>001</td></tr> <tr> <td>1/2</td><td>010</td></tr> <tr> <td>1/3</td><td>011</td></tr> <tr> <td>1/4</td><td>100</td></tr> </tbody> </table>	LPC_DIV	Bit Encoding	1/1	001	1/2	010	1/3	011	1/4	100
LPC_DIV	Bit Encoding										
1/1	001										
1/2	010										
1/3	011										
1/4	100										
NFC_DIV	NFC Divide Ratio <table border="1"> <thead> <tr> <th>NFC_DIV</th><th>Bit Encoding</th></tr> </thead> <tbody> <tr> <td>1/1</td><td>001</td></tr> <tr> <td>1/2</td><td>010</td></tr> <tr> <td>1/3</td><td>011</td></tr> <tr> <td>1/4</td><td>100</td></tr> </tbody> </table>	NFC_DIV	Bit Encoding	1/1	001	1/2	010	1/3	011	1/4	100
NFC_DIV	Bit Encoding										
1/1	001										
1/2	010										
1/3	011										
1/4	100										

Offset 0x0C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							IPS_DIV			PCI_DIV						MBX_GPX_DIV
W																
Reset	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MBX_GPX_DIV		LPC_DIV			NFC_DIV			DIU_DIV							
W																
Reset	0	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0

= Unimplemented or Reserved

Figure 5-12. System Clock Frequency Register 1 (SCFR1)

Table 5-9. SCFR1 Field Descriptions (Sheet 3 of 3)

Field	Description																												
DIU_DIV	DIU Divide Ratio <table border="1" data-bbox="678 1018 1133 1549"> <thead> <tr> <th>CSB_CLK:DIU_CLK</th><th>Field value</th></tr> </thead> <tbody> <tr><td>2</td><td>8</td></tr> <tr><td>3</td><td>12</td></tr> <tr><td>3.25</td><td>13</td></tr> <tr><td>3.5</td><td>14</td></tr> <tr><td>3.75</td><td>15</td></tr> <tr><td>4</td><td>16</td></tr> <tr><td>4.25</td><td>17</td></tr> <tr><td>N/4</td><td>N</td></tr> <tr><td>62.75</td><td>251</td></tr> <tr><td>63</td><td>252</td></tr> <tr><td>63.25</td><td>253</td></tr> <tr><td>63.5</td><td>254</td></tr> <tr><td>63.75</td><td>255</td></tr> </tbody> </table> <p>Note: The DIU Divide Ratio is related to the CSB_CLK. The DIU_CLK is derived directly from the SYS_CLK.</p>	CSB_CLK:DIU_CLK	Field value	2	8	3	12	3.25	13	3.5	14	3.75	15	4	16	4.25	17	N/4	N	62.75	251	63	252	63.25	253	63.5	254	63.75	255
CSB_CLK:DIU_CLK	Field value																												
2	8																												
3	12																												
3.25	13																												
3.5	14																												
3.75	15																												
4	16																												
4.25	17																												
N/4	N																												
62.75	251																												
63	252																												
63.25	253																												
63.5	254																												
63.75	255																												

5.3.1.5 System Clock Frequency Register 2 (SCFR2)

The system clock control register 2 shown in Figure 5-13 programs SYS_DIV ratio. Changes to this register value unlock the Power Architecture PLL, because the source clock of the Power Architecture PLL is derived from the SYS_CLK.

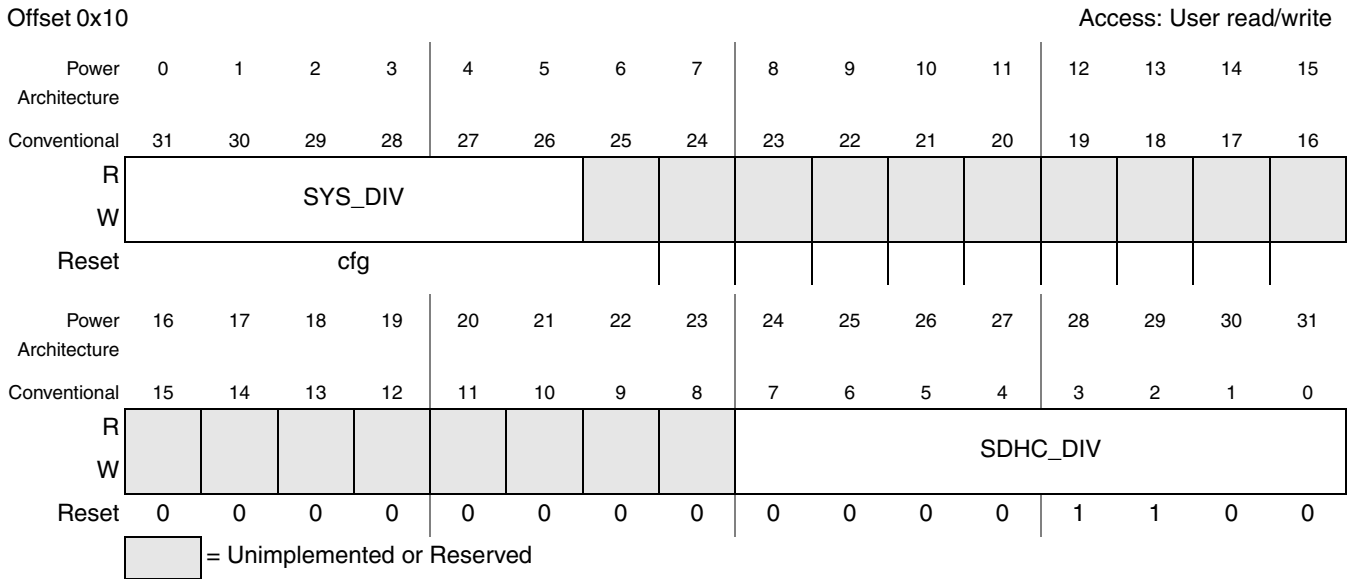


Figure 5-13. System Clock Frequency Register (SCFR2)

Table 5-10. SCFR2 Field Descriptions

Field	Description																																																																										
SYS_DIV	<div>SYS_CLK Divide Ratio</div> <div><table><thead><tr><th>Divide Factor</th><th>Bit Encoding</th></tr></thead><tbody><tr><td>2</td><td>000000</td></tr><tr><td>2.5</td><td>000001</td></tr><tr><td>3</td><td>000010</td></tr><tr><td>3.5</td><td>000011</td></tr><tr><td>4</td><td>000100</td></tr><tr><td>4.5</td><td>000101</td></tr><tr><td>5</td><td>000110</td></tr><tr><td>6</td><td>001000</td></tr><tr><td>7</td><td>000111</td></tr><tr><td>8</td><td>001001</td></tr><tr><td>9</td><td>001010</td></tr><tr><td>10</td><td>001100</td></tr><tr><td>11</td><td>001011</td></tr><tr><td>12</td><td>001101</td></tr><tr><td>13</td><td>001110</td></tr><tr><td>14</td><td>010000</td></tr><tr><td>15</td><td>001111</td></tr><tr><td>16</td><td>010001</td></tr></tbody></table><table><thead><tr><th>Divide Factor</th><th>Bit Encoding</th></tr></thead><tbody><tr><td>17</td><td>010010</td></tr><tr><td>18</td><td>010100</td></tr><tr><td>19</td><td>010011</td></tr><tr><td>20</td><td>010101</td></tr><tr><td>21</td><td>010110</td></tr><tr><td>22</td><td>011000</td></tr><tr><td>23</td><td>010111</td></tr><tr><td>24</td><td>011001</td></tr><tr><td>25</td><td>011010</td></tr><tr><td>26</td><td>011100</td></tr><tr><td>27</td><td>011011</td></tr><tr><td>28</td><td>011101</td></tr><tr><td>29</td><td>011110</td></tr><tr><td>30</td><td>100000</td></tr><tr><td>31</td><td>011111</td></tr><tr><td>32</td><td>100001</td></tr><tr><td>33</td><td>100010</td></tr></tbody></table><div>All other settings are reserved.</div></div>	Divide Factor	Bit Encoding	2	000000	2.5	000001	3	000010	3.5	000011	4	000100	4.5	000101	5	000110	6	001000	7	000111	8	001001	9	001010	10	001100	11	001011	12	001101	13	001110	14	010000	15	001111	16	010001	Divide Factor	Bit Encoding	17	010010	18	010100	19	010011	20	010101	21	010110	22	011000	23	010111	24	011001	25	011010	26	011100	27	011011	28	011101	29	011110	30	100000	31	011111	32	100001	33	100010
Divide Factor	Bit Encoding																																																																										
2	000000																																																																										
2.5	000001																																																																										
3	000010																																																																										
3.5	000011																																																																										
4	000100																																																																										
4.5	000101																																																																										
5	000110																																																																										
6	001000																																																																										
7	000111																																																																										
8	001001																																																																										
9	001010																																																																										
10	001100																																																																										
11	001011																																																																										
12	001101																																																																										
13	001110																																																																										
14	010000																																																																										
15	001111																																																																										
16	010001																																																																										
Divide Factor	Bit Encoding																																																																										
17	010010																																																																										
18	010100																																																																										
19	010011																																																																										
20	010101																																																																										
21	010110																																																																										
22	011000																																																																										
23	010111																																																																										
24	011001																																																																										
25	011010																																																																										
26	011100																																																																										
27	011011																																																																										
28	011101																																																																										
29	011110																																																																										
30	100000																																																																										
31	011111																																																																										
32	100001																																																																										
33	100010																																																																										
SDHC_DIV	<div>SDHC Divide Ratio</div> <div><table><thead><tr><th>CSB_CLK:SDHC_CLK</th><th>Field value</th></tr></thead><tbody><tr><td>2</td><td>8</td></tr><tr><td>3</td><td>12</td></tr><tr><td>3.25</td><td>13</td></tr><tr><td>3.5</td><td>14</td></tr><tr><td>3.75</td><td>15</td></tr><tr><td>4</td><td>16</td></tr><tr><td>4.25</td><td>17</td></tr><tr><td>N/4</td><td>N</td></tr><tr><td>62.75</td><td>251</td></tr><tr><td>63</td><td>252</td></tr><tr><td>63.25</td><td>253</td></tr><tr><td>63.5</td><td>254</td></tr><tr><td>63.75</td><td>255</td></tr></tbody></table></div>	CSB_CLK:SDHC_CLK	Field value	2	8	3	12	3.25	13	3.5	14	3.75	15	4	16	4.25	17	N/4	N	62.75	251	63	252	63.25	253	63.5	254	63.75	255																																														
CSB_CLK:SDHC_CLK	Field value																																																																										
2	8																																																																										
3	12																																																																										
3.25	13																																																																										
3.5	14																																																																										
3.75	15																																																																										
4	16																																																																										
4.25	17																																																																										
N/4	N																																																																										
62.75	251																																																																										
63	252																																																																										
63.25	253																																																																										
63.5	254																																																																										
63.75	255																																																																										

5.3.1.6 Shadow of System Clock Frequency Register 2 (SCFR2S)

The Shadow of System Clock Control Register 2 shown in [Figure 5-14](#), programs SYS_DIV ratio. When it gets control signal from PMC to update the main SCFR2, the value of this shadow register overwrites the SCFR2.

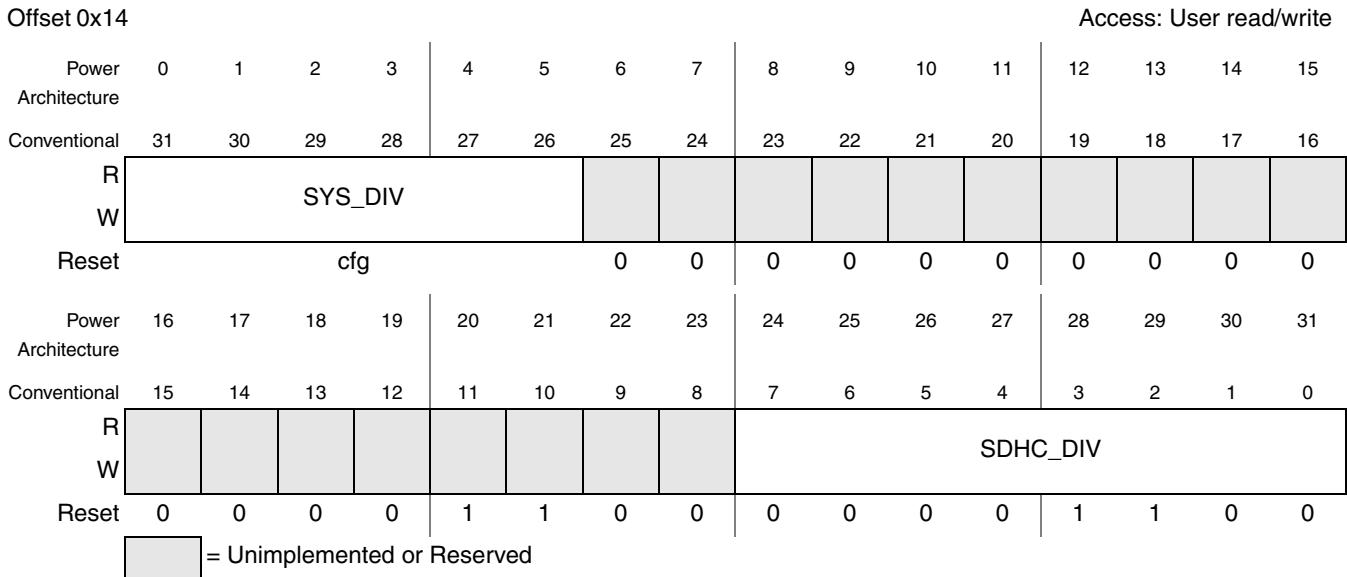


Figure 5-14. Shadow of System Clock Frequency Register (SCFR2S)
(Note that the register is repeated for reference.)

5.3.1.7 Bread Crumb Register (BCR)

The bread crumb register shown in [Figure 5-15](#) provides a mechanism for retaining data after reset. Data in this register is not affected by POR, HRESET, or SRESET.

Offset 0x18												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	N/A															
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	N/A															

Figure 5-15. Bread Crumb Register (BCR)

5.3.1.8 PSC0 Clock Control Register (P0CCR)

The PSC0 clock control register shown in Figure 5-16 controls the PSC0 MCLK divider ratio, PSC0 MCLK divider enable, PSC0 MCLK divider source, and the PSC0 MCLK source.

Table 5-11 defines the bit fields of P0CCR.

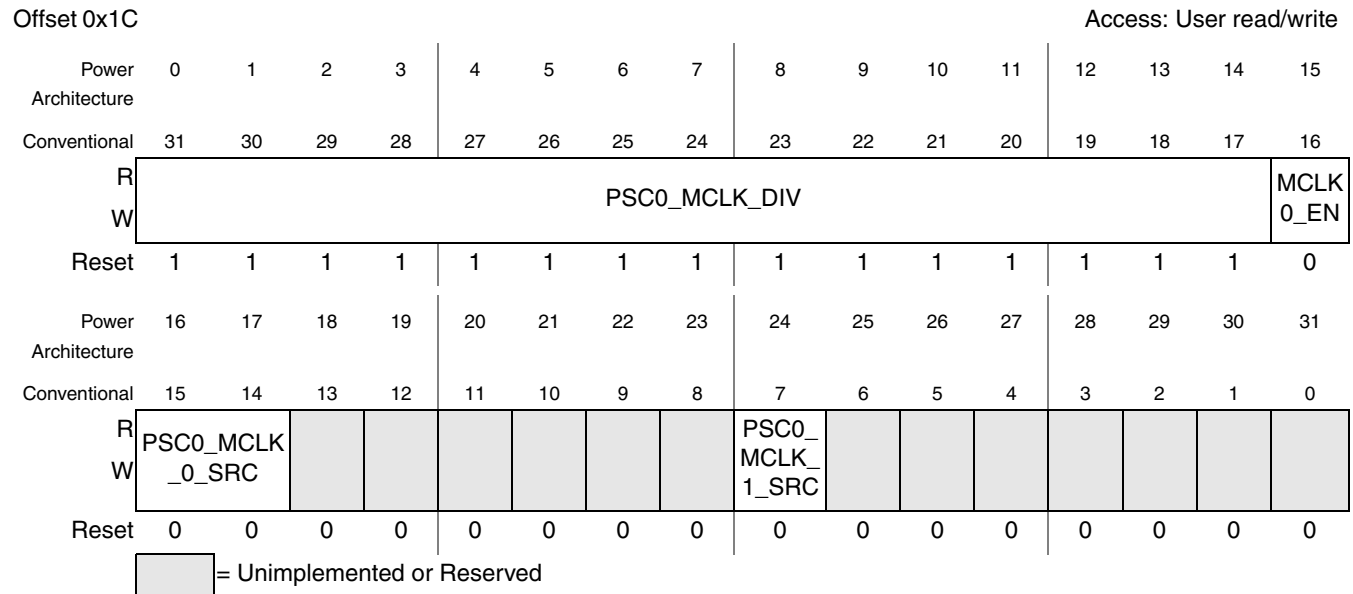


Figure 5-16. PSC0 Clock Control Register (P0CCR)

Table 5-11. P0CCR Field Descriptions

Field	Description
PSC0_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN = 0
MCLK0_EN	PSC0 Divider Enable 0 PSC0 divider is disable 1 PSC0 divider is enable
PSC0_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC0_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.9 PSC1 Clock Control Register (P1CCR)

The PSC1 clock control register shown in Figure 5-17 controls the PSC1 MCLK divider ratio, PSC1 MCLK divider enable, PSC1 MCLK divider source, and the PSC1 MCLK source.

Table 5-12 defines the bit fields of P1CCR.

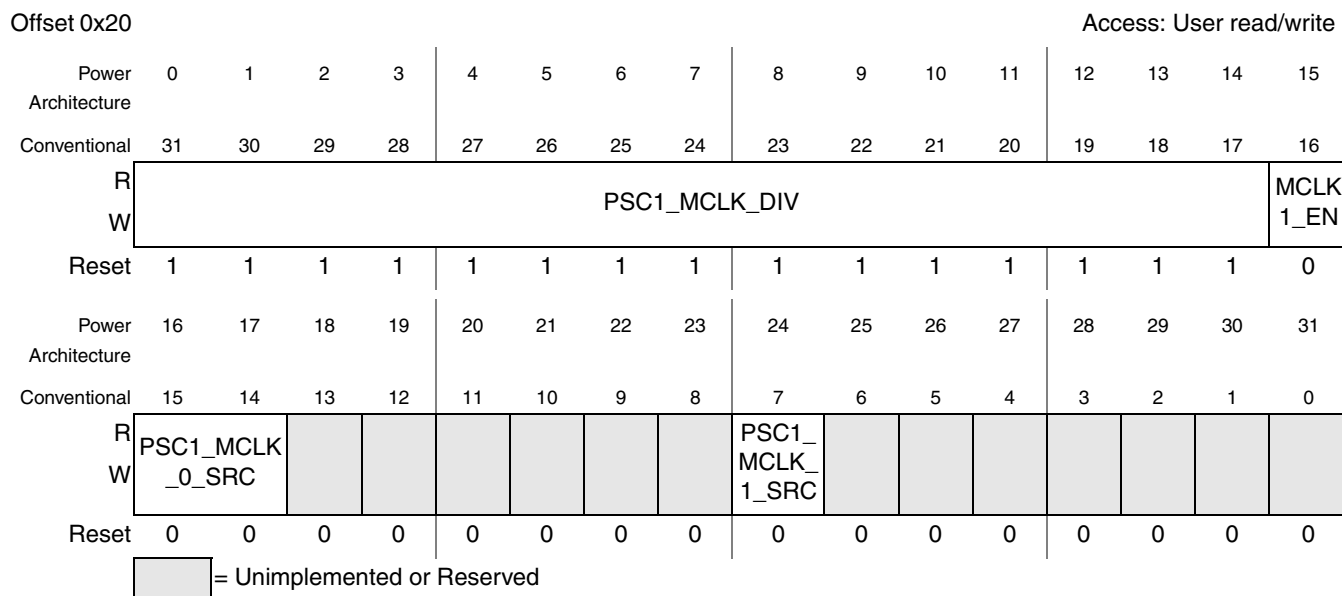


Figure 5-17. PSC1 Clock Control Register (P1CCR)

Table 5-12. P1CCR Field Descriptions

Field	Description
PSC1_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK1_EN	PSC1 Divider Enable 0 PSC1 divider is disable 1 PSC1 divider is enable
PSC1_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC1_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.10 PSC2 Clock Control Register (P2CCR)

The PSC2 clock control register shown in Figure 5-18 controls the PSC2 MCLK divider ratio, PSC2 MCLK divider enable, PSC2 MCLK divider source, and the PSC2 MCLK source.

Table 5-13 defines the bit fields of P2CCR.

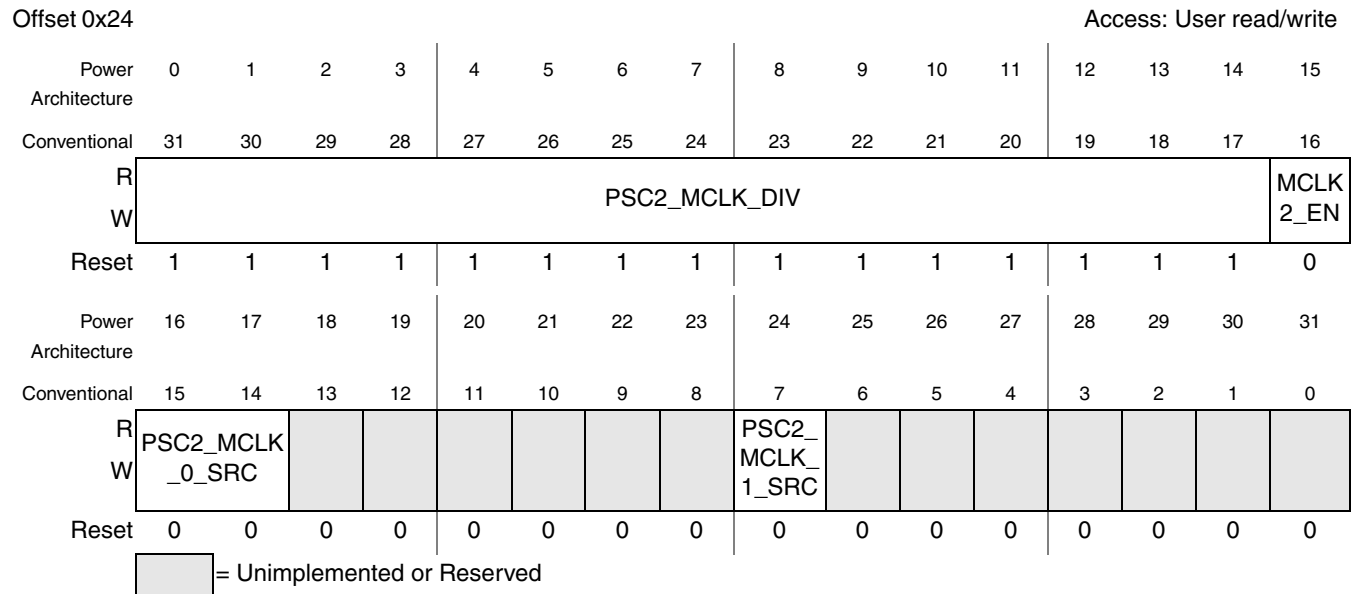


Figure 5-18. PSC2 Clock Control Register (P2CCR)

Table 5-13. P2CCR Field Descriptions

Field	Description
PSC2_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK2_EN	PSC2 Divider Enable 0 PSC2 divider is disable 1 PSC2 divider is enable
PSC2_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC2_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.11 PSC3 Clock Control Register (P3CCR)

The PSC3 clock control register shown in Figure 5-19 controls the PSC3 MCLK divider ratio, PSC3 MCLK divider enable, PSC3 MCLK divider source, and the PSC3 MCLK source.

Table 5-14 defines the bit fields of P3CCR.

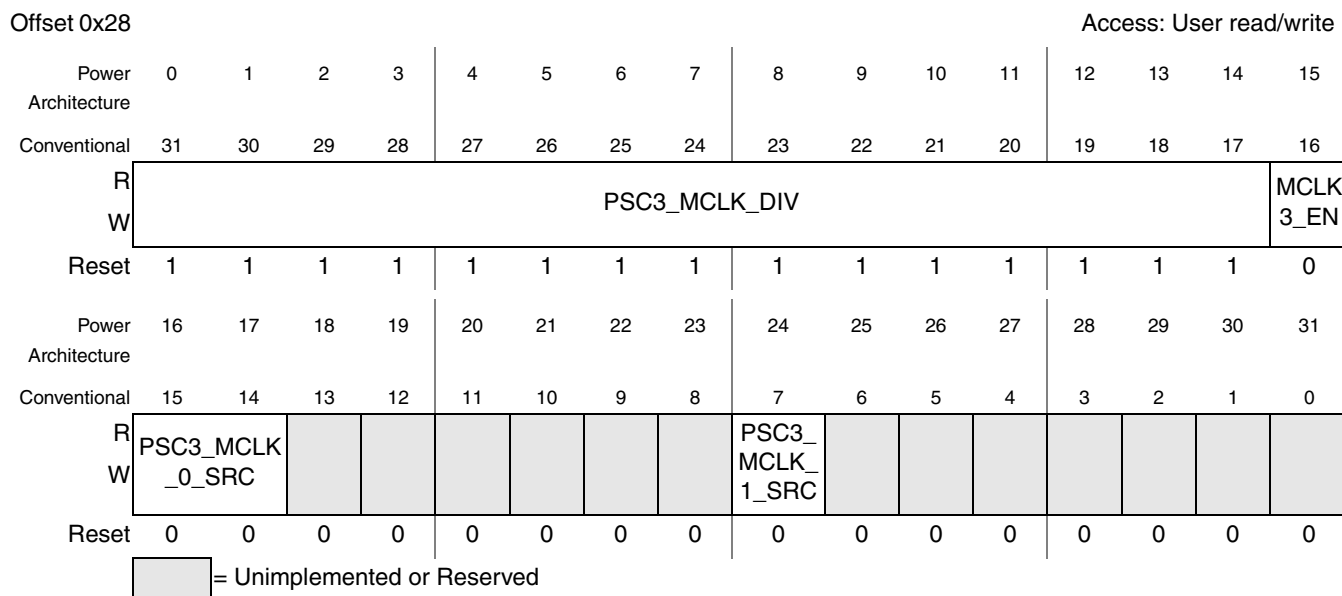


Figure 5-19. PSC3 Clock Control Register (P3CCR)

Table 5-14. P3CCR Field Descriptions

Field	Description
PSC3_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK3_EN	PSC3 Divider Enable 0 PSC3 divider is disable 1 PSC3 divider is enable
PSC3_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC3_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.12 PSC4 Clock Control Register (P4CCR)

The PSC4 clock control register shown in Figure 5-20 controls the PSC4 MCLK divider ratio, PSC4 MCLK divider enable, PSC4 MCLK divider source, and the PSC4 MCLK source.

Table 5-15 defines the bit fields of P4CCR.

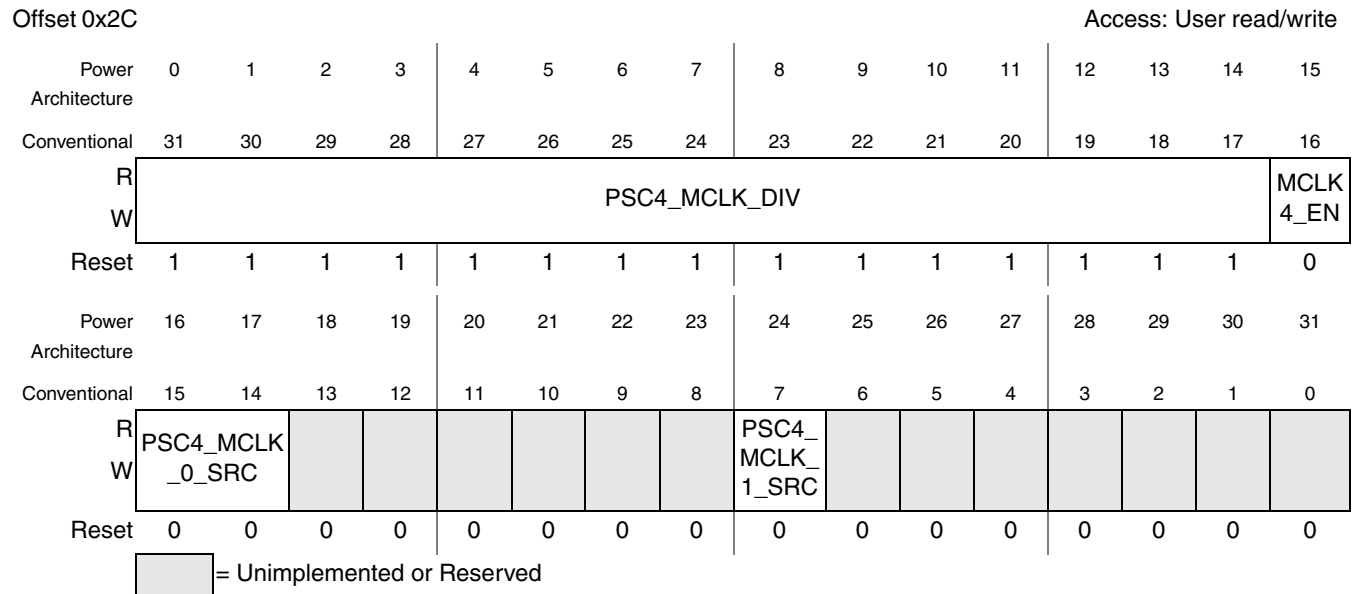


Figure 5-20. PSC4 Clock Control Register (P4CCR)

Table 5-15. P4CCR Field Descriptions

Field	Description
PSC4_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK4_EN	PSC4 Divider Enable 0 PSC4 divider is disable 1 PSC4 divider is enable
PSC4_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC4_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.13 PSC5 Clock Control Register (P5CCR)

The PSC5 clock control register shown in Figure 5-21 controls the PSC5 MCLK divider ratio, PSC5 MCLK divider enable, PSC5 MCLK divider source, and the PSC5 MCLK source.

Table 5-16 defines the bit fields of P5CCR.

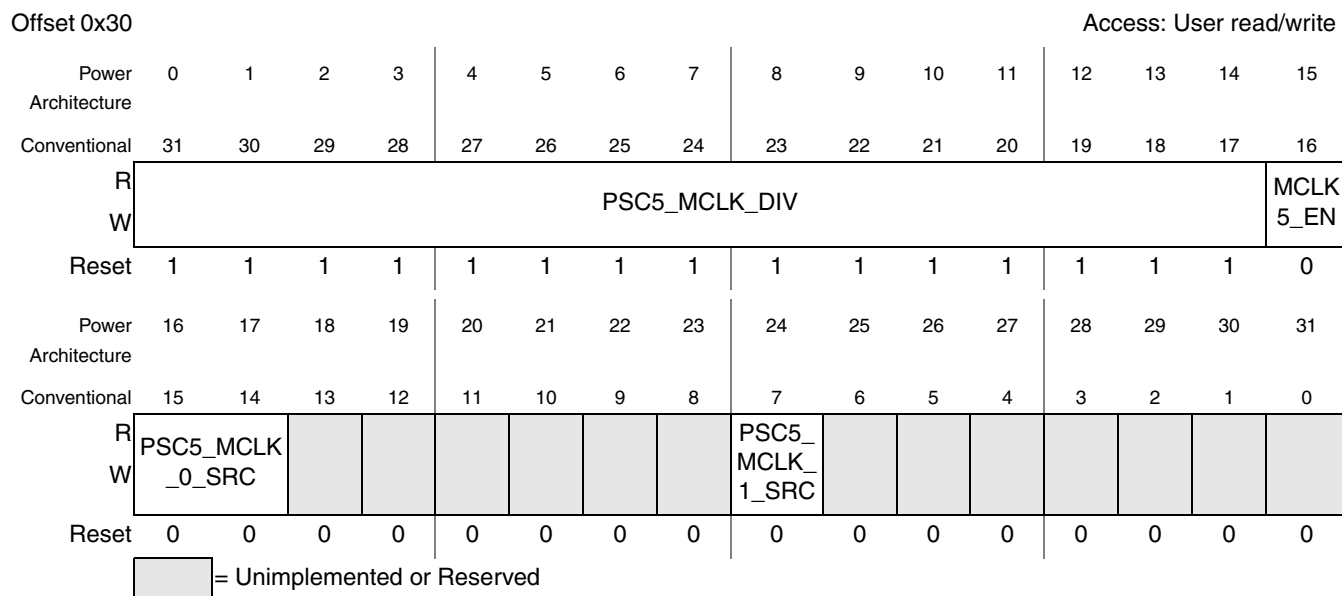


Figure 5-21. PSC5 Clock Control Register (P5CCR)

Table 5-16. P5CCR Field Descriptions

Field	Description
PSC5_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK5_EN	PSC5 Divider Enable 0 PSC5 divider is disable 1 PSC5 divider is enable
PSC5_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC5_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.14 PSC6 Clock Control Register (P6CCR)

The PSC6 clock control register shown in Figure 5-22 controls the PSC6 MCLK divider ratio, PSC6 MCLK divider enable, PSC6 MCLK divider source, and the PSC6 MCLK source.

Table 5-17 defines the bit fields of P6CCR.

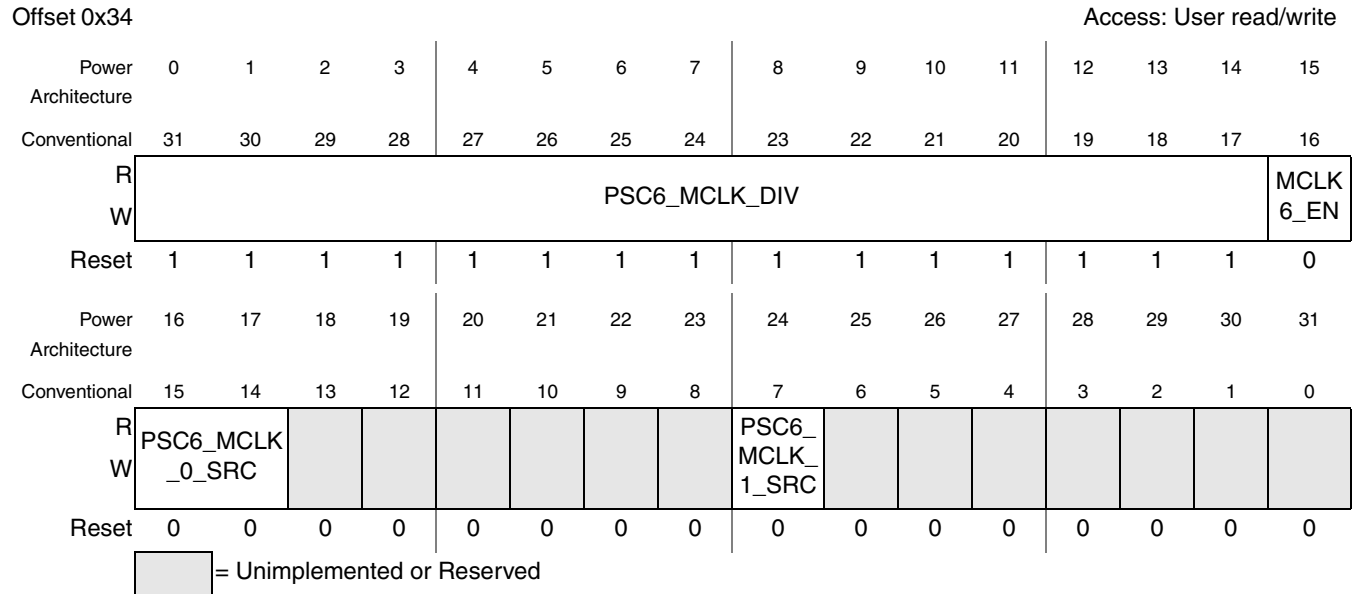


Figure 5-22. PSC6 Clock Control Register (P6CCR)

Table 5-17. P6CCR Field Descriptions

Field	Description
PSC6_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK6_EN	PSC Divider Enable 0 PSC6 divider is disable 1 PSC6 divider is enable
PSC6_MCLK_0_SRC	PSC MCLK Divider Source 00 From SYS_CLK 01 FROM REF_CLK 10 FROM PSC_MCLK_IN 11 FROM SPDIF_TXCLK
PSC6_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.15 PSC7 Clock Control Register (P7CCR)

The PSC7 clock control register shown in Figure 5-23 controls the PSC7 MCLK divider ratio, PSC7 MCLK divider enable, PSC7 MCLK divider source, and the PSC7 MCLK source.

Table 5-18 defines the bit fields of P7CCR.

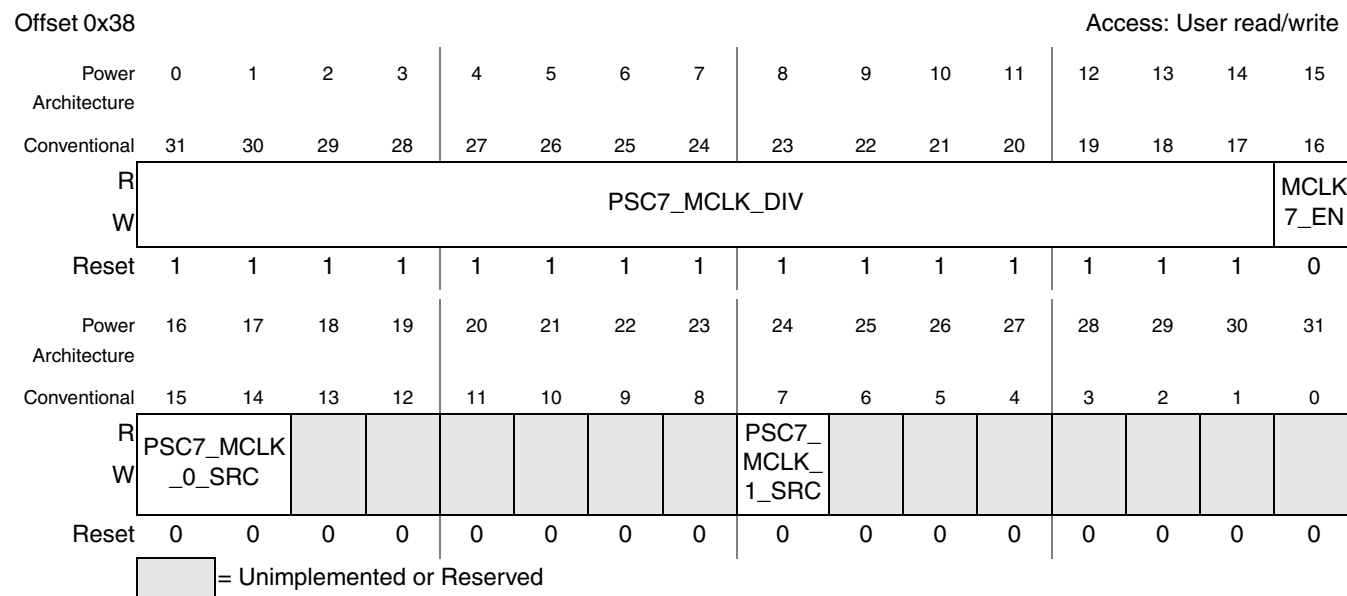


Figure 5-23. PSC7 Clock Control Register (P7CCR)

Table 5-18. P7CCR Field Descriptions

Field	Description
PSC7_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN = 0
MCLK7_EN	PSC7 Divider Enable 0 PSC7 divider is disable 1 PSC7 divider is enable
PSC7_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC7_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.16 PSC8 Clock Control Register (P8CCR)

The PSC8 clock control register shown in Figure 5-24 controls the PSC8 MCLK divider ratio, PSC8 MCLK divider enable, PSC8 MCLK divider source, and the PSC8 MCLK source. Table 5-19 defines the bit fields of P8CCR.

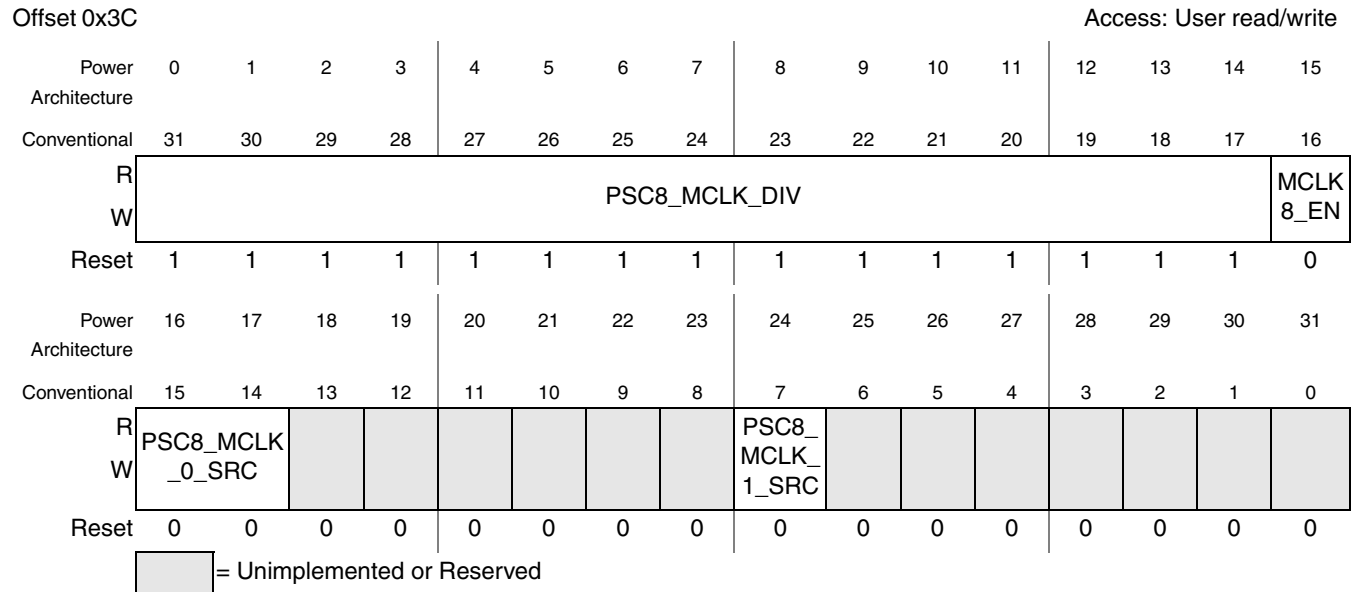


Figure 5-24. PSC8 Clock Control Register (P8CCR)

Table 5-19. P8CCR Field Descriptions

Field	Description
PSC8_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK8_EN	PSC8 Divider Enable 0 PSC8 divider is disable 1 PSC8 divider is enable
PSC8_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC8_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.17 PSC9 Clock Control Register (P9CCR)

The PSC9 clock control register shown in Figure 5-25 controls the PSC9 MCLK divider ratio, PSC9 MCLK divider enable, PSC9 MCLK divider source, and the PSC9 MCLK source.

Table 5-20 defines the bit fields of P9CCR.

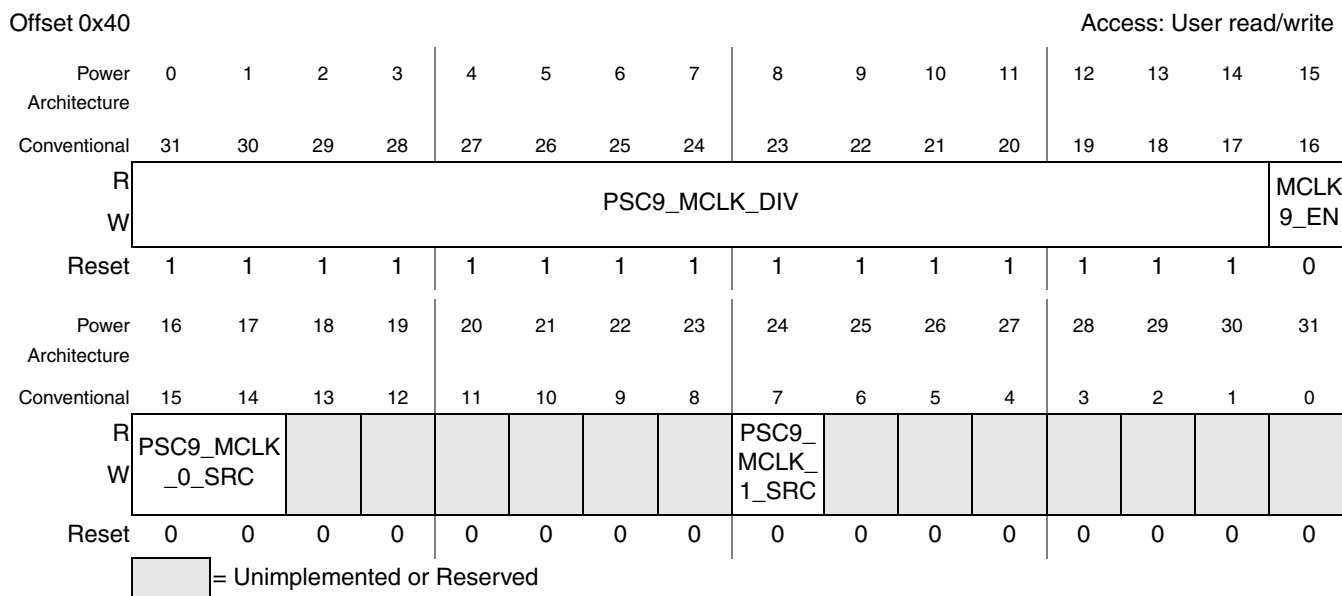


Figure 5-25. PSC9 Clock Control Register 9 (P9CCR)

Table 5-20. P9CCR Field Descriptions

Field	Description
PSC9_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK9_EN	PSC9 Divider Enable 0 PSC9 divider is disable 1 PSC9 divider is enable
PSC9_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC9_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.18 PSC10 Clock Control Register (P10CCR)

The PSC10 clock control register shown in Figure 5-26 controls the PSC10 MCLK divider ratio, PSC10 MCLK divider enable, PSC10 MCLK divider source, and the PSC10 MCLK source.

Table 5-21 defines the bit fields of P10CCR.

Offset 0x44

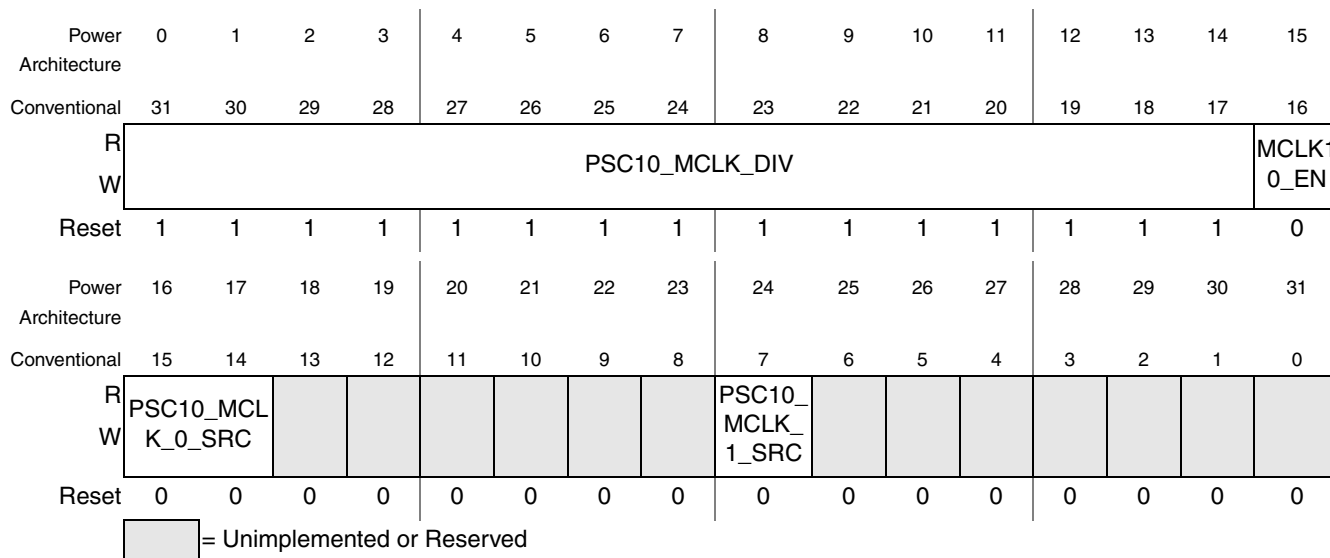


Figure 5-26. PSC10 Clock Control Register 10 (P10CCR)

Table 5-21. P10CCR Field Descriptions

Field	Description
PSC10_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK10_EN	PSC10 Divider Enable 0 PSC10 divider is disable 1 PSC10 divider is enable
PSC10_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC10_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.19 PSC11 Clock Control Register (P11CCR)

The PSC11 clock control register shown in Figure 5-27 controls the PSC11 MCLK divider ratio, PSC11 MCLK divider enable, PSC11 MCLK divider source, and the PSC11 MCLK source.

Table 5-22 defines the bit fields of P11CCR.

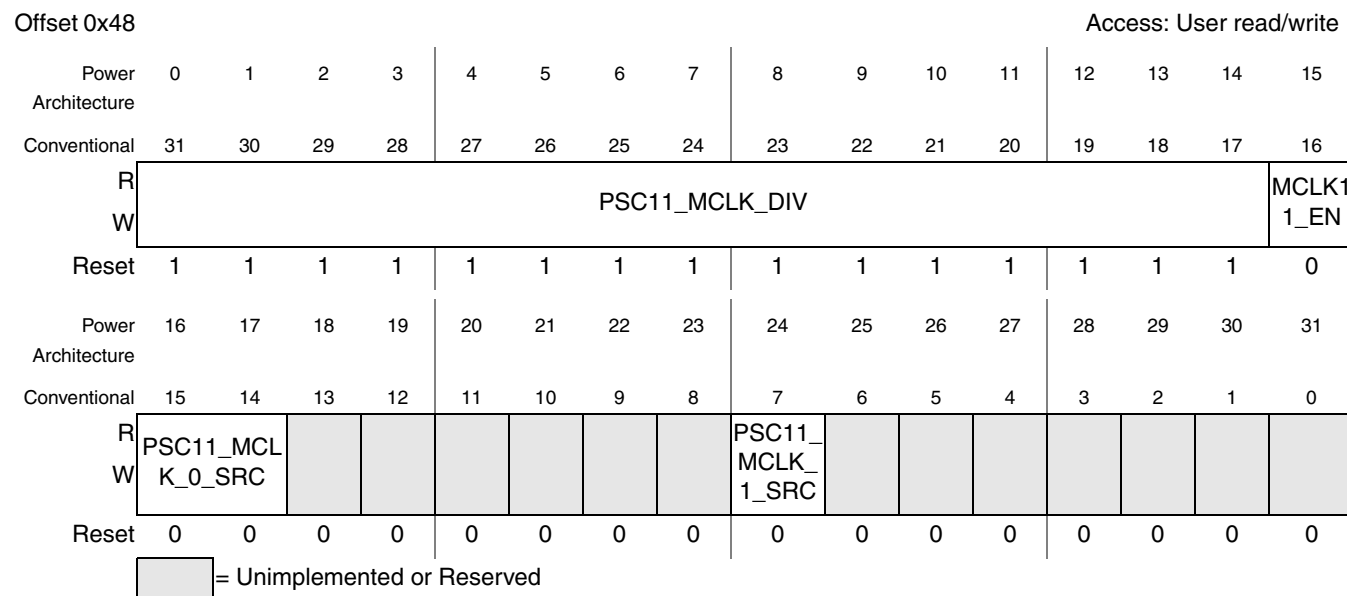


Figure 5-27. PSC11 Clock Control Register 11 (P11CCR)

Table 5-22. P11CCR Field Descriptions

Field	Description
PSC11_MCLK_DIV	MCLK_DIV divider Ratio $f_{mclk_out} = f_{mclk_src} / (MCLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MCLK_EN equals 0
MCLK11_EN	PSC11 Divider Enable 0 PSC11 divider is disable 1 PSC11 divider is enable
PSC11_MCLK_0_SRC	PSC Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
PSC11_MCLK_1_SRC	PSC MCLK Source 0 MCLK_DIV 1 SPDIF_RXCLK

5.3.1.20 SPDIF Clock Control Register (SCCR)

The SPDIF clock control register shown in Figure 5-28 controls the SPDIF MCLK divider ratio, SPDIF MCLK divider enable, SPDIF MCLK divider source, and the SPDIF MCLK source.

Table 5-23 defines the bit fields of SPCR.

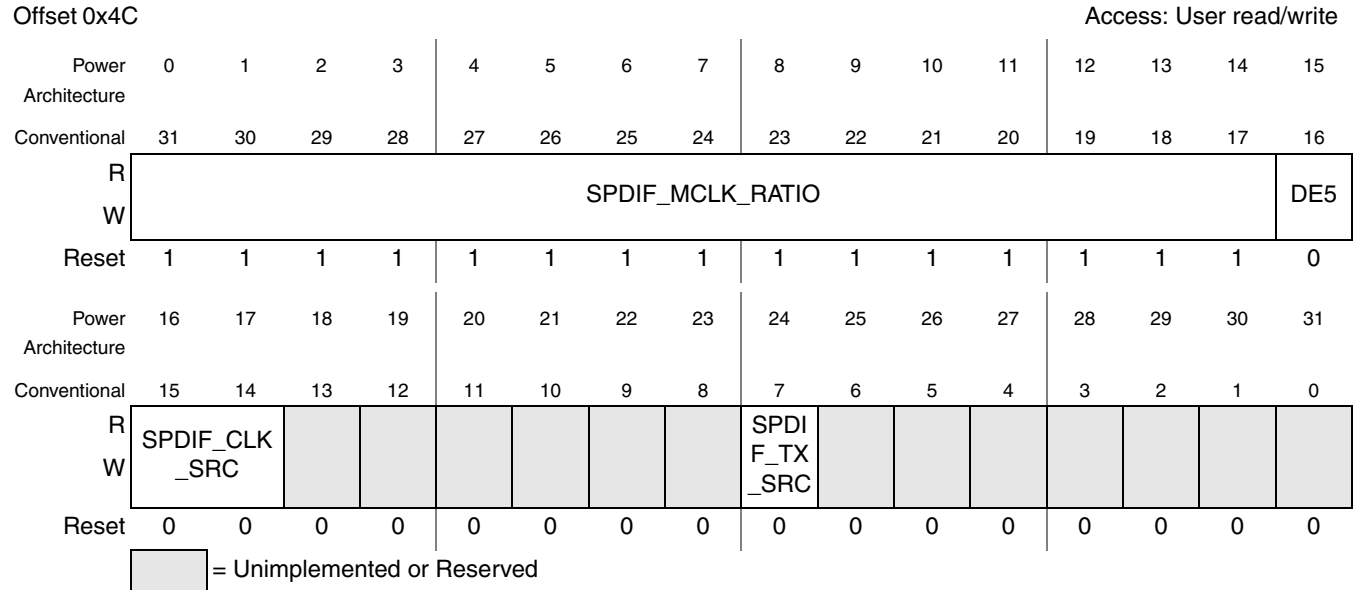


Figure 5-28. SPDIF Clock Control Register (SCCR)

Table 5-23. SCCR Field Descriptions

Field	Description
SPDIF_MCLK_RATIO	SPDIF Mclk Divider Ratio The Divided Clock Frequency = System PLL Clock Frequency/(SPDIF_CLK_RATIO +1) Note: 000_0000_0000_0000 is reserved. Note: The ratio change needs to be under divider disable state.
DE5	SPDIF Mclk Divider Enable 0 mclk divider 0 disable 1 mclk divider 0 enable
SPDIF_CLK_SRC	SPDIF Mclk Divider Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK
SPDIF_TX_SRC	SPDIF MCLK Source 0 SPDIF Clock Divider 1 SPDIF_RXCLK

5.3.1.21 CFM Clock Control Register (CCCR)

The CFM clock control register shown in [Figure 5-29](#) controls:

- CFM clock0 source
- CFM clock1 source
- CFM clock2 source
- CFM clock3 source

[Table 5-24](#) defines the bit fields of CCCR.

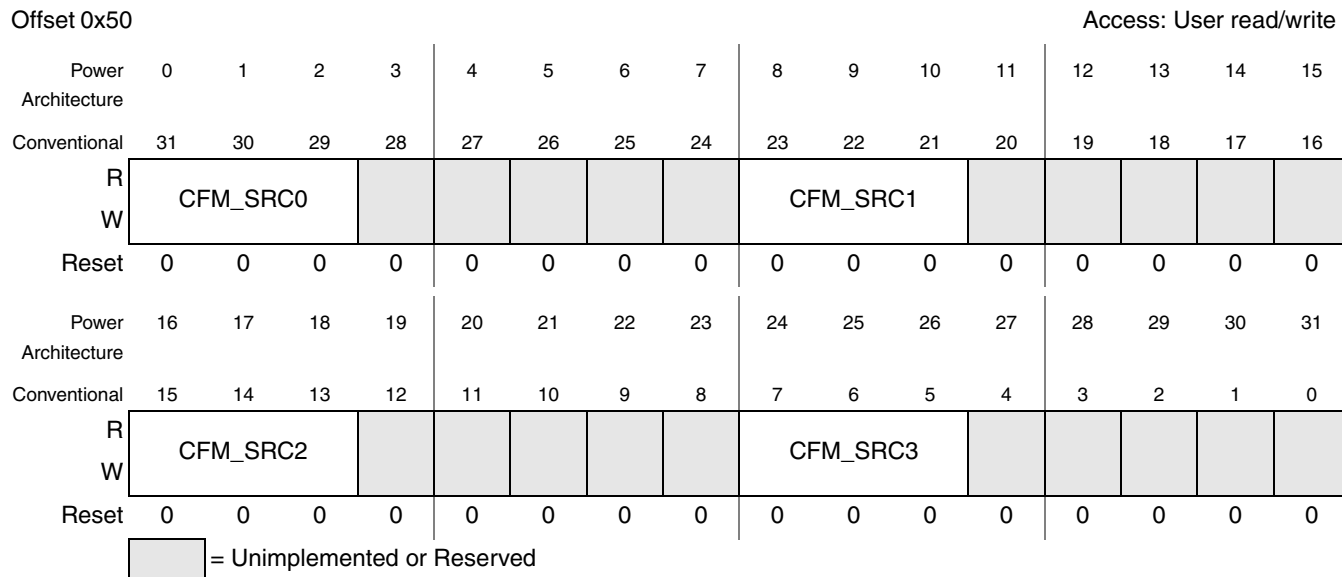


Figure 5-29. CFM Clock Control Register (CCCR)
(Register is repeated for reference.)

Table 5-24. CCCR Field Descriptions

Field	Description
CFM_SRC0	CFM Clock 0 Source Select 000 psc0_mclk_out 001 psc1_mclk_out 010 psc2_mclk_out 100 psc0_bclk 101 psc1_bclk 110 psc2_bclk
CFM_SRC1	CFM Clock 1 Source Select 000 psc3_mclk_out 001 psc4_mclk_out 010 psc5_mclk_out 100 psc3_bclk 101 psc4_bclk 110 psc5_bclk

Offset 0x50

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CFM_SRC0								CFM_SRC1							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CFM_SRC2								CFM_SRC3							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 5-29. CFM Clock Control Register (CCCR)
(Register is repeated for reference.)

Table 5-24. CCCR Field Descriptions (continued)

Field	Description
CFM_SRC2	CFM Clock 2 Source Select 000 psc6_mclk_out 001 psc7_mclk_out 010 psc8_mclk_out 100 psc6_bclk 101 psc7_bclk 110 psc8_bclk
CFM_SRC3	CFM Clock 3 Source Select 000 psc9_mclk_out 001 psc10_mclk_out 010 psc11_mclk_out 100 psc9_bclk 101 psc10_bclk 110 psc11_bclk

5.3.1.22 DIU Clock Config Register (DCCR)

The DIU clock config register shown in Figure 5-30 configures the number of CSB cycles delay added to pixel clock to pad compare to pixel clock to DIU block.

Table 5-25 defines the bit fields of DCCR.

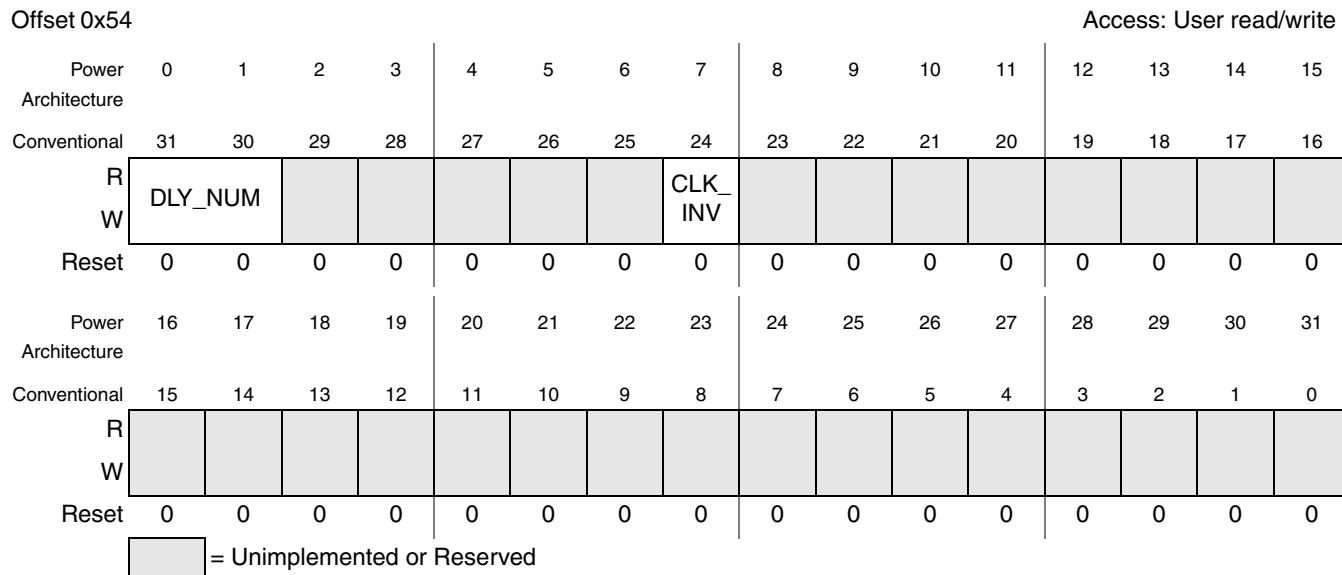


Figure 5-30. DIU Clock Config Register (DCCR)

Table 5-25. DCCR Field Descriptions

Field	Description
DLY_NUM	Number of CSB_CLK cycles delay added to pixel clock output to pad compare pixel clock to DIU. 00 0 cycle delay 01 2 cycle delay 10 4 cycles delay 11 6 cycles delay
CLK_INV	Pixel Clock Inversion 0 The pixel clock to pad is the same as the one to DIU. DLY_NUM decides the delay CSB cycles to it. 1 The pixel clock to pad is the inverted of the one to DIU. DLY_NUM decides the delay CSB cycles added to it.

5.3.1.23 MSCAN1Clock Control Register (M1CCR)

The MSCAN1 clock control register shown in Figure 5-31 controls the MSCAN1 source clock divider ratio, MSCAN1 divider enable and MSCAN1 divider clock source.

Table 5-26 defines the bit fields of M1CCR.

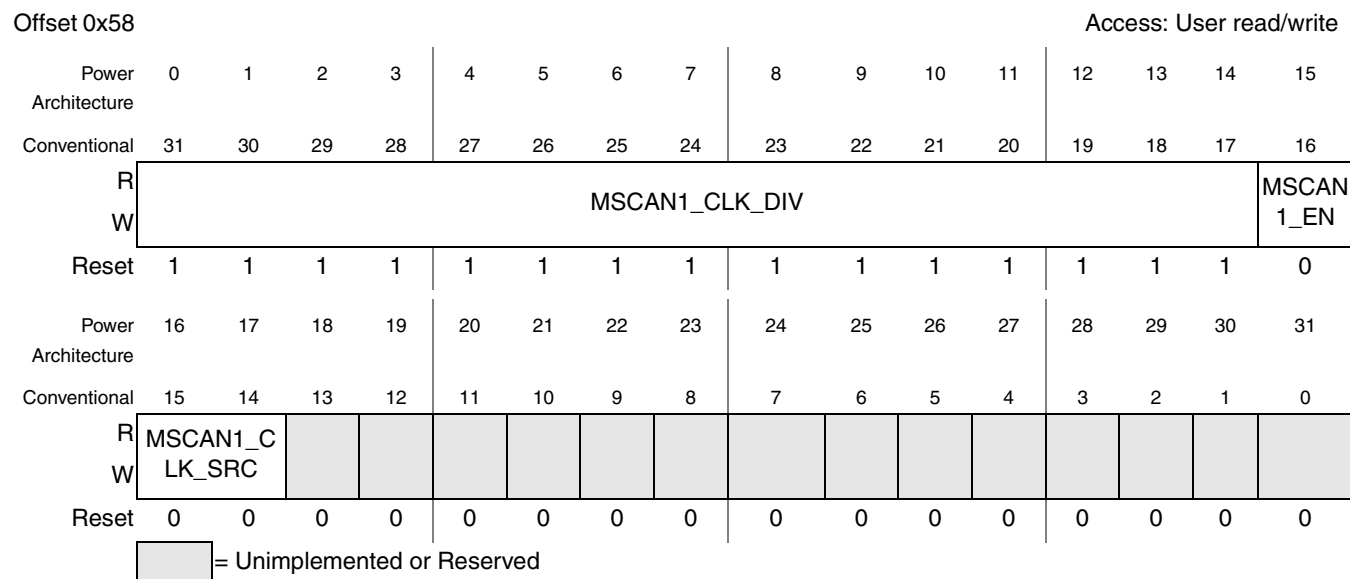


Figure 5-31. MSCAN1 Clock Control Register (M1CCR)

Table 5-26. M1CCR Field Descriptions

Field	Description
MSCAN1_CLK_DIV	MSCAN1 divider Ratio $f_{can_source_clk} = f_{mcan_src} / (MSCAN1_CLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MSCAN1_EN equals 0
MSCAN1_EN	MSCAN1 Divider Enable 0 MSCAN1 divider is disable 1 MSCAN1 divider is enable
MSCAN1_CLK_SRC	MSCAN1 CLK Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK

5.3.1.24 MSCAN2 Clock Control Register (M2CCR)

The MSCAN2 clock control register shown in Figure 5-32 controls the MSCAN2 source clock divider ratio, MSCAN2 divider enable, and MSCAN2 divider clock source.

Table 5-27 defines the bit fields of M2CCR.

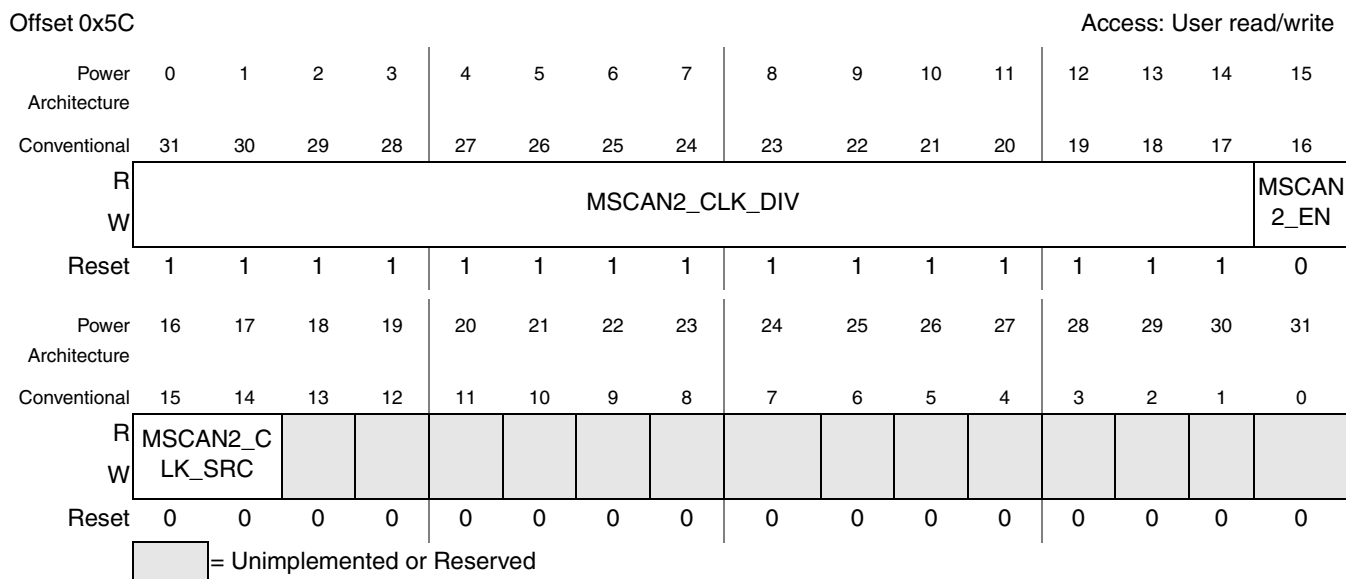


Figure 5-32. MSCAN2 Clock Control Register (M2CCR)

Table 5-27. M2CCR Field Descriptions

Field	Description
MSCAN2_CLK_DIV	MSCAN2 divider Ratio $f_{can_source_clk} = f_{mcan_src} / (MSCAN2_CLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MSCAN2_EN equals 0
MSCAN2_EN	MSCAN2 Divider Enable 0 MSCAN2 divider is disable 1 MSCAN2 divider is enable
MSCAN2_CLK_SRC	MSCAN2 CLK Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK

5.3.1.25 MSCAN3 Clock Control Register (M3CCR)

The MSCAN3 clock control register shown in Figure 5-33 controls the MSCAN3 source clock divider ratio, MSCAN3 divider enable and MSCAN3 divider clock source.

Table 5-28 defines the bit fields of M3CCR.

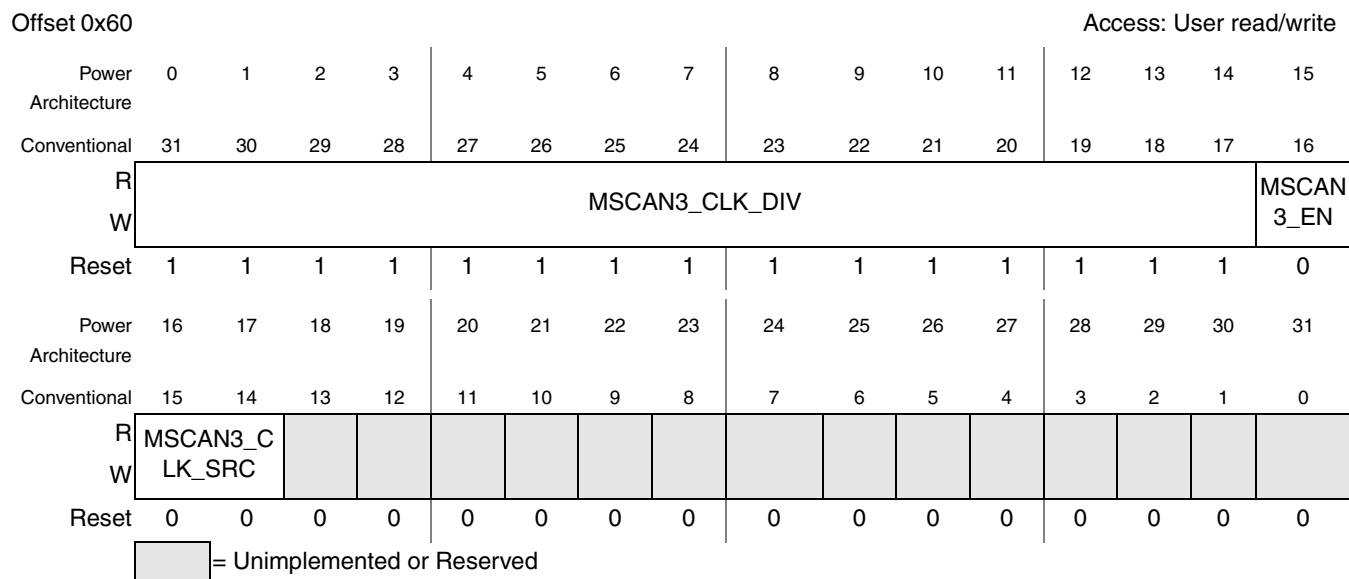


Figure 5-33. MSCAN3 Clock Control Register (M3CCR)

Table 5-28. M3CCR Field Descriptions

Field	Description
MSCAN3_CLK_DIV	MSCAN3 divider Ratio $f_{can_source_clk} = f_{mcan_src} / (MSCAN3_CLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MSCAN3_EN equals 0
MSCAN3_EN	MSCAN3 Divider Enable 0 MSCAN3 divider is disable 1 MSCAN3 divider is enable
MSCAN3_CLK_SRC	MSCAN3 CLK Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK

5.3.1.26 MSCAN4 Clock Control Register (M4CCR)

The MSCAN4 clock control register shown in Figure 5-34 controls the MSCAN4 source clock divider ratio, MSCAN4 divider enable and MSCAN4 divider clock source.

Table 5-29 defines the bit fields of M4CCR.

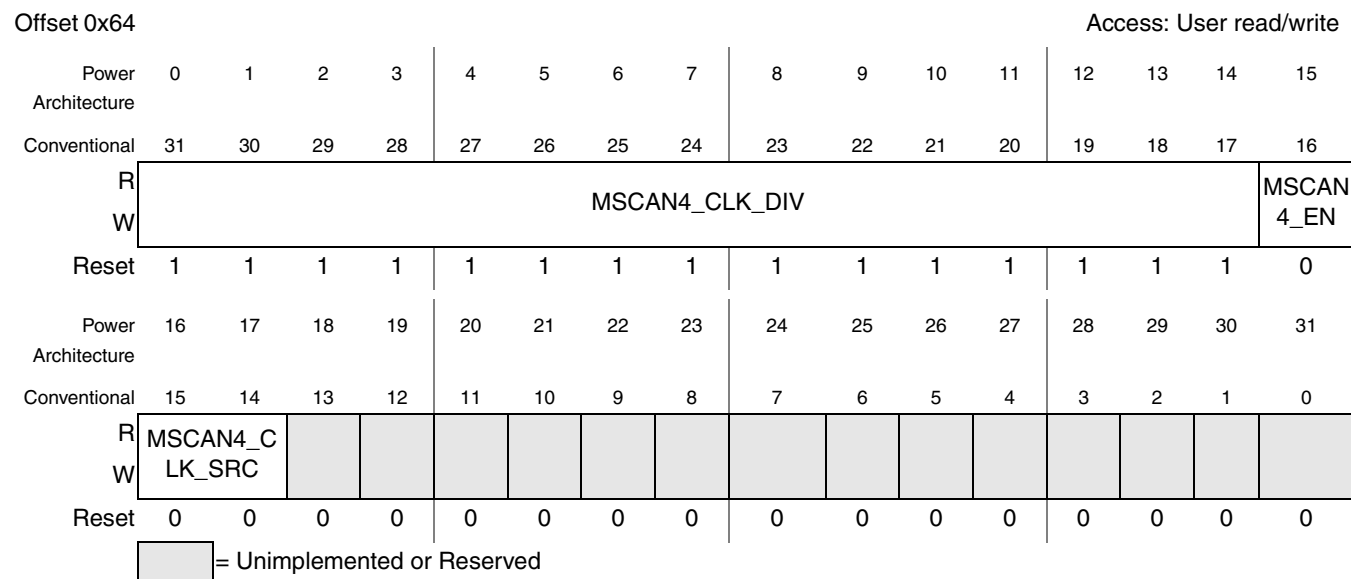


Figure 5-34. MSCAN4 Clock Control Register (M4CCR)

Table 5-29. M4CCR Field Descriptions

Field	Description
MSCAN4_CLK_DIV	MSCAN4 divider Ratio $f_{can_source_clk} = f_{mcan_src} / (MSCAN4_CLK_DIV + 1)$ A value of 0x0 bypass the divider Note: This value can only be changed when the value of MSCAN4_EN equals 0
MSCAN4_EN	MSCAN4 Divider Enable 0 MSCAN4 divider is disable 1 MSCAN4 divider is enable
MSCAN4_CLK_SRC	MSCAN4 CLK Source 00 From SYS_CLK 01 From REF_CLK 10 From PSC_MCLK_IN 11 From SPDIF_TXCLK

Chapter 6

AXE System

6.1 Introduction

Figure 6-1 shows a block diagram of the AXE system.

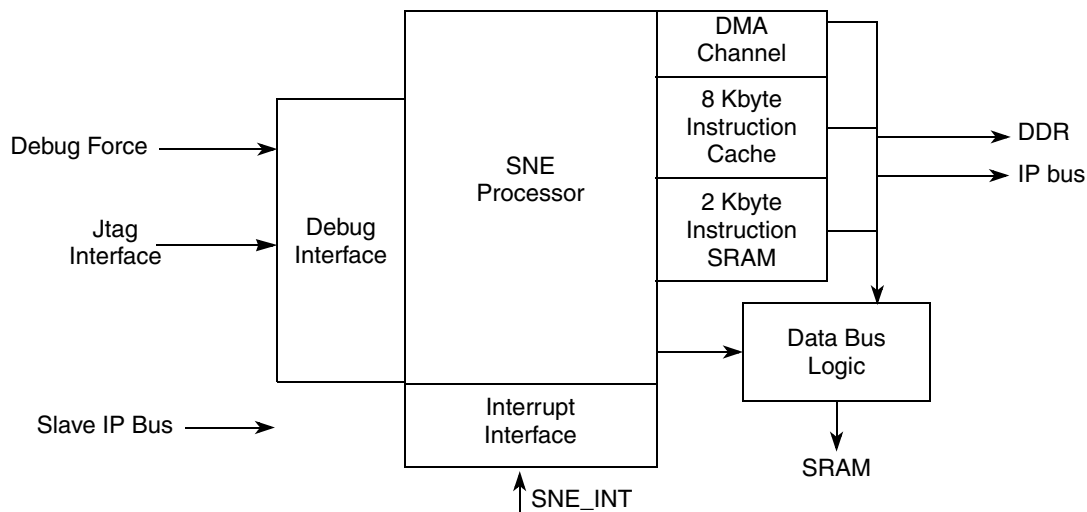


Figure 6-1. Block Diagram of the Auxiliary Execution Engine

The AXE system features a 200 MHz RISC microprocessor with two computation units: a scalar integer unit implementing a general 32-bit instruction set and a 48-bit fixed-point processing unit. The AXE is intended to off-load the main system CPU on compute and data-flow intensive operations (for example: digital signal processing and compressed audio encode/decode). This chapter provides a basic overview of features, operation, and system interfaces. Refer to the *AXE Reference Manual* for a detailed description of the core registers and instruction set.

6.1.1 Features

- SNE processor
 - 200 MHz clock
 - can execute scalar and vector instructions
 - Eight 32-bit data registers
 - Eight 32-bit address registers
 - Eight 48-bit vector register
 - Single-cycle 32*32 bit multiply and multiply/accumulate

- Sleep mode via register bit. Writing 1 to it sends the processor into low-power sleep mode. Any non-masked interrupt wakes the processor out of sleep mode.
- 8-Kbyte 1-way set-associative instruction cache
 - Feature to invalidate cache content
 - Possible to disable the cache
- 2-Kbyte instruction SRAM
- Single channel DMA for transfer of data between MEM and DRAM or IP Bus. The DMA works in 32-byte bursts, and allows up to four pipelined transfers. It is capable of transfer speeds in excess of 500 Mbyte/sec.
- Hardware debugger
 - One hardware program counter breakpoint register
 - One hardware address and data breakpoint register
 - Core has two modes: normal run and debug
 - Possible to enter debug mode via breakpoint or via debug instruction
 - In debug mode, it's possible to see all processor registers and peek into processor memory.
 - Hardware debugger is controlled via JTAG tap controller, compliant with IEEE 1149.1,
 - The debug breakpoint registers are accessible by the core. This allows dynamic breakpoint generation.
- Interrupt controller
 - There is an interrupt controller that is part of the core. It supports 32 interrupts. Every interrupt has a programmable 6-bit vector code. There is a priority encoder, detecting highest level of pending interrupt.
- Two 4x32 bit FIFOs for interprocessor communication

6.2 Memory Map and Register Definition

6.2.1 Data Memory Map

Table 6-1. Data Bus Memory Map

Offset or Address	Address Space	Access
Programmable ¹	On-Chip SRAM (MEM)	R/W
0 to 0x1FFF	AXE register space	R/W
Programmable ²	DDR-SDRAM	R/W
Other ³	IP Bus	R/W

¹ All addresses where bits [31:18] are equal to bits [31:18] of SRAM Base Address Register (SRAMBAR) are mapped to the on-chip SRAM

² All addresses which are falling into the DDR address window, specified by DDR Local Access Window Base Address Register (DDRLAWBAR) and DDR Local Access Window Attributes Register (DDRLAWAR) within XLBMEN memory map, are mapped to the DDR-SDRAM.

³ All addresses that do not hit in any of the three above are mapped to the IP Bus

Table 6-2. IP-Bus Memory Map

Offset or Address	Register	Access
0x00	DMA Address Register	R/W
0x04	DMA MEM Address Register	R/W
0x08	DMA Attributes Register	R/W
0x0C	Instruction Cache and SRAM Attributes Register	R/W
0x10	INTPRI0 Interrupt Priority Register	R/W
0x14	INTPRI1 Interrupt Priority Register	R/W
0x18	INTPRI2 Interrupt Priority Register	R/W
0x1C	INTPRI3 Interrupt Priority Register	R/W
0x20	INTPRI4 Interrupt Priority Register	R/W
0x24	INTPRI5 Interrupt Priority Register	R/W
0x28	INTPRI6 Interrupt Priority Register	R/W
0x2C	INTPRI7 Interrupt Priority Register	R/W
0x30	Instruction SRAM Address Register	R/W
0x34	Instruction SRAM Data Register	R/W
0x3C	FIFO1 Write Data Register	W
0x40	FIFO1 Read Data Register	R
0x44	FIFO2 Write Data Register	W
0x48	FIFO2 Read Data Register	R
0x4C	FIFO Level Register	R
0x50	SNE Interrupt Enable Register	RW
0x54	Power Architecture Interrupt Enable Register	RW
0x58	Interrupt Clear/Set Register	W
0x5C	Interrupt Pending Register	R
0x60	Instruction Cache Address Register	RW
0x64	Instruction Cache Data Register	RW
0x68	Instruction Cache Tag Register	RW

6.2.2 Instruction Memory Map

Table 6-3. Instruction Bus Memory Map

Offset or Address	Address Space	Access
Programmable ¹	On-Chip SRAM (MEM)	R/W
0 to 0x1FFF	AXE instruction SRAM	R/W

Table 6-3. Instruction Bus Memory Map

Offset or Address	Address Space	Access
Programmable ²	DDR-SDRAM	R/W
Other ³	IP Bus	R/W

¹ All addresses where bits [31:18] are equal to bits [31:18] of SRAM Base Address Register (SRAMBAR) are mapped to the on-chip SRAM

² All addresses which are falling into the DDR address window, specified by DDR Local Access Window Base Address Register (DDRLAWBAR) and DDR Local Access Window Attributes Register (DDRLAWAR) within XLBMEN memory map, are mapped to the DDR-SDRAM.

³ All addresses that do not hit in any of the three above are mapped to the IP Bus

6.2.3 Register Summary

Table 6-4 shows the format for a register summary table

Table 6-4. AXE Register Summary (Sheet 1 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
DMA_ADDR 0x00	R																			
	W																			
	R																			
	W																			
DMA_MEM_ADDR 0x04	R													DMA_MEM_ADDR[19:16]						
	W																			
	R																			
	W																			
DMA_ATTRIBUTE S 0x08	R												DMA_WRITE	INTEN		INTPEND	DMA_ON			
	W														INTCLEAR					
	R																			
	W																			
INSTRUCTION SRAM, CACHE ATTRIBUTE S 0x0C	R									SRAM2 DUSERON	SRAM2 DSUPON	SRAM2 IUSERON	SRAM2 ISUPON	CACHEON		IRAM_USERON	IRAM_SUPON	PROC RST		
	W														CACHECLEAR					
	R																			
	W																			

Table 6-4. AXE Register Summary (Sheet 2 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTPRI0 0x10	R		INT0 ON	INT0 PRI[5:0]							INT1 ON	INT1 PRI[5:0]					
	W																
	R		INT2 ON	INT2 PRI[5:0]							INT3 ON	INT3 PRI[5:0]					
	W																
INTPRI1 0x14	R		INT4 ON	INT4 PRI[5:0]							INT5 ON	INT5 PRI[5:0]					
	W																
	R		INT6 ON	INT6 PRI[5:0]							INT7 ON	INT7 PRI[5:0]					
	W																
INTPRI2 0x18	R		INT8 ON	INT8 PRI[5:0]							INT9 ON	INT9 PRI[5:0]					
	W																
	R		INT10 ON	INT10 PRI[5:0]							INT11 ON	INT11 PRI[5:0]					
	W																
INTPRI3 0x1C	R		INT12 ON	INT12 PRI[5:0]							INT13 ON	INT13 PRI[5:0]					
	W																
	R		INT14 ON	INT14 PRI[5:0]							INT15 ON	INT15 PRI[5:0]					
	W																
INTPRI4 0x20	R		INT16 ON	INT16 PRI[5:0]							INT17 ON	INT17 PRI[5:0]					
	W																
	R		INT18 ON	INT18 PRI[5:0]							INT19 ON	INT19 PRI[5:0]					
	W																
INTPRI5 0x24	R		INT20 ON	INT20 PRI[5:0]							INT21 ON	INT21 PRI[5:0]					
	W																
	R		INT22 ON	INT22 PRI[5:0]							INT23 ON	INT23 PRI[5:0]					
	W																
INTPRI6 0x28	R		INT24 ON	INT24 PRI[5:0]							INT25 ON	INT25 PRI[5:0]					
	W																
	R		INT26 ON	INT26 PRI[5:0]							INT27 ON	INT27 PRI[5:0]					
	W																
INTPRI7 0x2C	R		INT28 ON	INT28 PRI[5:0]							INT29 ON	INT29 PRI[5:0]					
	W																
	R		INT30 ON	INT30 PRI[5:0]							INT31 ON	INT31 PRI[5:0]					
	W																

Table 6-4. AXE Register Summary (Sheet 3 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INST SRAM ADDRESS 0x30	R																
	W																
	R	INSTR_ADDR[15:2]															
	W																
INSTR SRAM DATA 0x34	R	INSTR_DATA[31:16]															
	W																
	R	INSTR_DATA[15:0]															
	W																
FIFO1_WDATA TA 0x3C	R																
	W	FIFO1_WDATA[31:16]															
	R																
	W	FIFO1_WDATA[15:0]															
FIFO1_RDATA A 0x40	R	FIFO1_RDATA[31:16]															
	W																
	R	FIFO1_RDATA[15:0]															
	W																
FIFO2_WDATA TA 0x44	R																
	W	FIFO2_WDATA[31:16]															
	R																
	W	FIFO2_WDATA[15:0]															
FIFO2_RDATA A 0x48	R	FIFO2_RDATA[31:16]															
	W																
	R	FIFO2_RDATA[15:0]															
	W																
FIFO_LEVEL 0x4C	R						FIFO1_FILL[2:0]								FIFO2_FILL[2:0]		
	W																
	R																
	W																
SNE_INTEN 0x50	R	SNE_SOFTINT[7:0]									SF1EE	SF1NEE	SF1NFE	SF1UE	SF1OE		
	W																
	R	SF2EE	SF2NEE	SF2NFE	SF2UE	SF2OE											
	W																

Table 6-4. AXE Register Summary (Sheet 4 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPC_INTEN 0x54	R	PPC SOFTINT[7:0]								PF1EE	PF1NEE	PF1NFE	PF1UE	PF1OE			
	W																
	R	PF2EE	PF2NEE	PF2NFE	PF2UE	PF2OE											
	W																
INTCLEARS ET 0x58	R																
	W	SOFTINT_SET[7:0]								SOFTINT_CLEAR[7:0]							
	R																
	W												F1UC	F1OC	F2UC	F2OC	
INT_PENDIN G 0x5C	R	SOFTINT[7:0]								F1EP	F1NEP	F1NFP	F1UP	F1OP			
	W																
	R	F2EP	F2NEP	F2NFP	F2UP	F2OP											
	W																
ICACHE_ADDR 0x60	R																
	W																
	R				ICACHE_ADDR[12:2]												
	W																
ICACHE_DATA 0x64	R	ICACHE_DATA[31:16]															
	W																
	R	ICACHE_DATA[15:0]															
	W																
ICACHE_TAG 0x68	R												V	S	Addr[31:29]		
	W																
	R	Addr[28:13]															
	W																

6.2.4 Register Descriptions

6.2.4.1 DMA Operation

6.2.4.1.1 DMA Address Register

Offset 0x00Access: User read/write

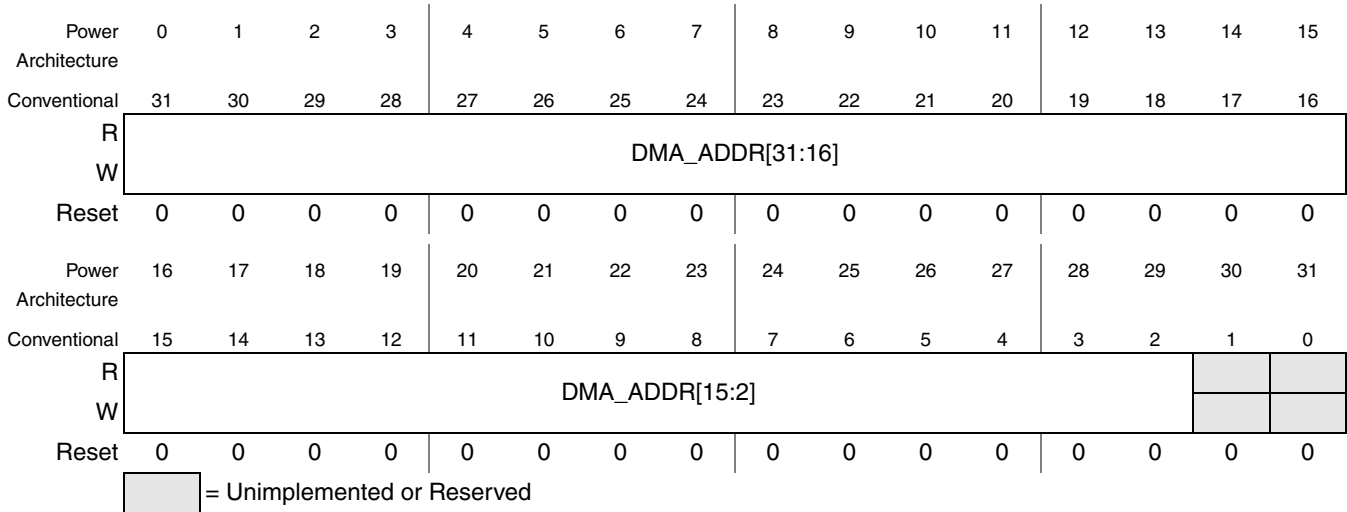


Figure 6-2. DMA Address Register

Table 6-5. DMA Address Register Field Descriptions

Field	Description
DMA_ADDR [31:2]	Contains the first or next address the DMA uses to address DDR or IP Bus. It is auto-incremented by the DMA. Note: Do not write this register while the DMA is running. It is valid to read this register when the DMA is running, but it changes as it is auto-incremented every time a DMA access to DDR or IP Bus is done.

6.2.4.1.2 DMA MEM Address Register

Offset 0x04 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													DMA_MEM_ADDR[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMA_MEM_ADDR[15:2]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-3. DMA MEM Address Register

Table 6-6. DMA MEM Address Register Field Descriptions

Field	Description
DMA_MEM_ADDR	<p>Contains the first or next address the DMA uses to address the on-chip SRAM (MEM). It is auto-incremented by the DMA.</p> <p>Note: Do not write this register while the DMA is running. It is valid to read this register when the DMA is running, but it changes as it is auto-incremented every time a DMA access to MEM is done.</p>

6.2.4.1.3 DMA Attributes Register

Offset 0x08 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R												DMA_DIRECTION	INT_EN		INT_PENDING	DMA_ON
W														INT_CLEAR		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WORD_COUNT[13:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-4. DMA Attributes Register

Table 6-7. DMA Attributes Register Field Descriptions

Field	Description
DMA_DIRECTION	1: DMA transfers from the on-chip SRAM (MEM) to the DDR or IP Bus 0: DMA transfers from the DDR or IP Bus to the on-chip SRAM (MEM)
INT_EN	0: DMA finish event doesn't generate an interrupt to the SNE processor 1: DMA finish event generates an interrupt to the SNE processor
INT_CLEAR	Writing 1 to this bit clears interrupt pending bit.
INT_PENDING	Interrupt pending. This bit is set when DMA finishes the current transfer.
DMA_ON	DMA on bit. If 1, the DMA is armed and transfers data until WORD_COUNT decrements to 0.
WORD_COUNT	The DMA transfer count in 32-words. It is auto-decremented by the DMA. Note: Do not write while the DMA is running, or indeterminate operation results. It is valid to read while the DMA is running, but the count is updated every time data is transferred

6.2.4.2 Instruction Cache and SRAM Operation

6.2.4.2.1 Instruction Cache and SRAM Attributes Register

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							SLEEP	SRAM 2 DUSE R ON	SRAM 2 DSUP ON	SRAM 2 IUSE R ON	SRAM 2 ISUP ON	CACH E ON		IRAM _USE R_ON	IRAM _SUP _ON	PROC RST
W													CACH E CLEAR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0
	= Unimplemented or Reserved															

Figure 6-5. Instruction Cache and SRAM Attributes Register

Table 6-8. Instruction Cache and SRAM Attributes Register Field Descriptions

Field	Description
SLEEP	1: AXE is in sleep mode, all clocks stopped 0: AXE is in run mode, normal operation.
SRAM2 DUSER ON	1: Data accesses to on-chip SRAM (MEM) are enabled in SNE user mode 0: Data accesses to on-chip SRAM (MEM) are disabled in SNE user mode
SRAM2 DSUP ON	1: Data accesses to on-chip SRAM (MEM) are enabled in SNE supervisor mode 0: Data accesses to on-chip SRAM (MEM) are disabled in SNE supervisor mode
SRAM2 IUSER ON	1: Instruction accesses to on-chip SRAM (MEM) are enabled in SNE user mode 0: Instruction accesses to on-chip SRAM (MEM) are disabled in SNE user mode
SRAM2 ISUP ON	1: Instruction accesses to on-chip SRAM (MEM) are enabled in SNE supervisor mode 0: Instruction accesses to on-chip SRAM (MEM) are disabled in SNE supervisor mode
CACHE ON	1: Instruction cache is enabled 0: Instruction cache is disabled
CACHE CLEAR	Writing a 1 to this bit, invalidates the instruction cache
IRAM_USER_ON	1: Instruction accesses to instruction SRAM are enabled in SNE user mode 0: Instruction accesses to instruction SRAM are disabled in SNE user mode
IRAM_SUP_ON	1: Instruction accesses to instruction SRAM are enabled in SNE supervisor mode 0: Instruction accesses to instruction SRAM are disabled in SNE supervisor mode
PROC RST	Processor Reset 1: SNE processor kept in reset, not executing code 0: SNE processor is executing code

AXE enters sleep mode when sleep mode is written 1. Wake up happens when any of following occurs:

- The Power Architecture processor writes a zero to the SLEEP bit, instructing the processor to run again.
- An interrupt, that is not disabled, is made pending to the SNE processor.

6.2.4.3 Interrupt Controller

6.2.4.3.1 INTPRIO Interrupt Priority Register

Offset 0x10 Access: User read/write

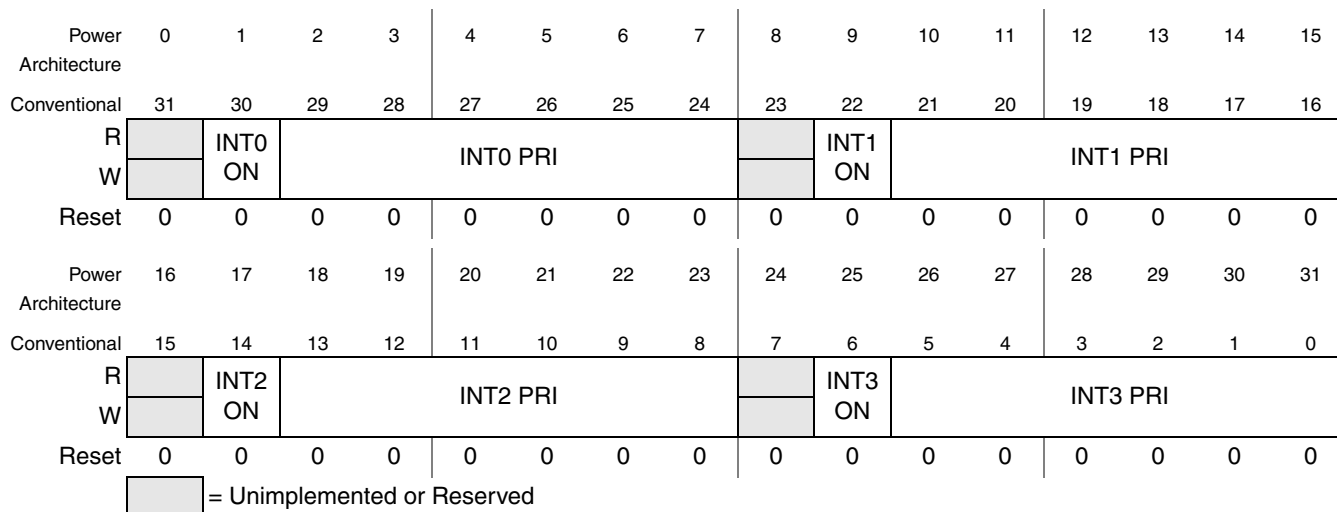


Figure 6-6. INTPRIO Interrupt Priority Register

Table 6-9. INTPRIO Interrupt Priority Register Field Descriptions

Field	Description
INT0 ON	1: Interrupt 0 enable. Pending interrupt 0 interrupts SNE processor 0: Interrupt 0 disable. Pending interrupt 0 does not interrupt SNE processor
INT0 PRI	Interrupt 0 priority and interrupt vector. Note: Interrupt priority 0 is the highest priority. Interrupt priority 63 is the lowest priority.
INT1 ON - INT3 ON	Same as INT0 ON, but for interrupt 1 - 3
INT1 PRI - INT3 PRI	Same as INT0 PRI, but for interrupt 1 - 3

6.2.4.3.2 INTPRI1 Interrupt Priority Register

Offset 0x14 Access: User read/write

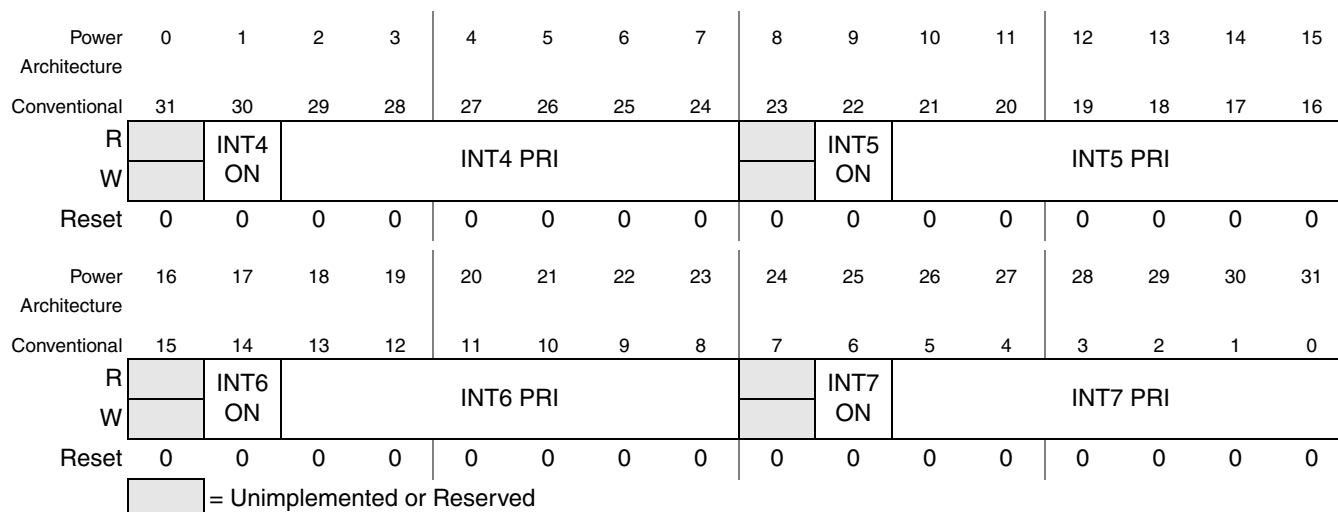


Figure 6-7. INTPRI1 Interrupt Priority Register

Table 6-10. INTPRI1 Interrupt Priority Register Field Descriptions

Field	Description
INT4 ON - INT7 ON	Same as INT0 ON, but for interrupt 4 - 7
INT4 PRI - INT7 PRI	Same as INT0 PRI, but for interrupt 4 - 7

6.2.4.3.3 INTPRI2 Interrupt Priority Register

Offset 0x18Access: User read/write

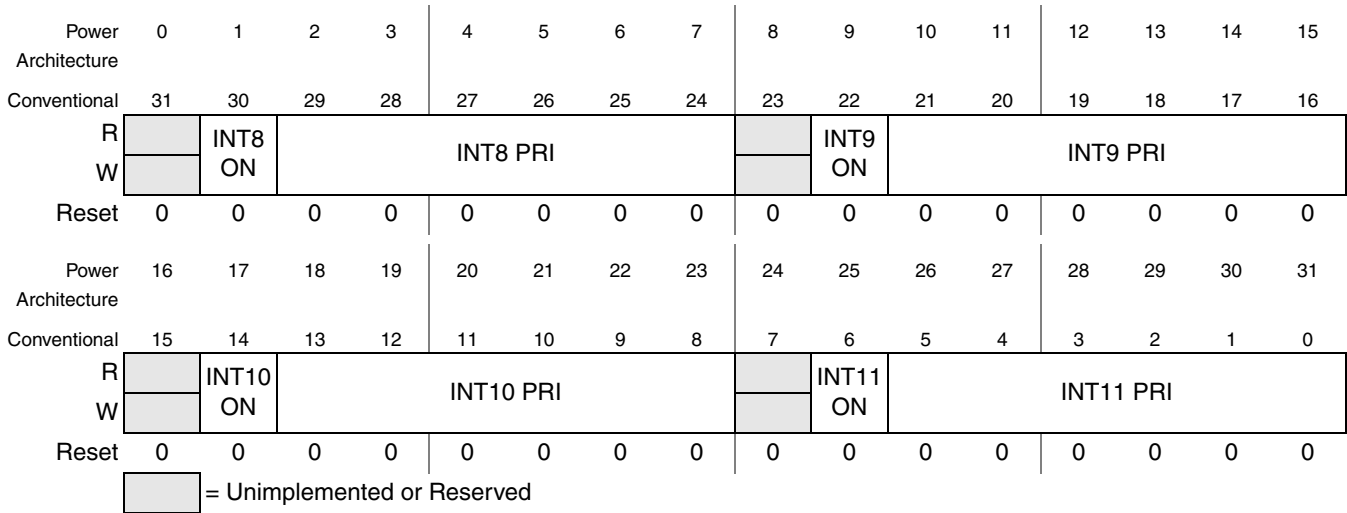


Figure 6-8. INTPRI2 Interrupt Priority Register

Table 6-11. INTPRI2 Interrupt Priority Register Field Descriptions

Field	Description
INT8 ON - INT11 ON	Same as INT0 ON, but for interrupt 8 - 11
INT8 PRI - INT11 PRI	Same as INT0 PRI, but for interrupt 8 - 11

6.2.4.3.4 INTPRI3 Interrupt Priority Register

Offset 0x1C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		INT12 ON	INT12 PRI							INT13 ON	INT13 PRI					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		INT14 ON	INT14 PRI							INT15 ON	INT15 PRI					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-9. INTPRI3 Interrupt Priority Register

Table 6-12. INTPRI3 Interrupt Priority Register Field Descriptions

Field	Description
INT12 ON - INT15 ON	Same as INT0 ON, but for interrupt 12 - 15
INT12 PRI - INT15 PRI	Same as INT0 PRI, but for interrupt 12 - 15

6.2.4.3.5 INTPRI4 Interrupt Priority Register

Offset 0x20Access: User read/write

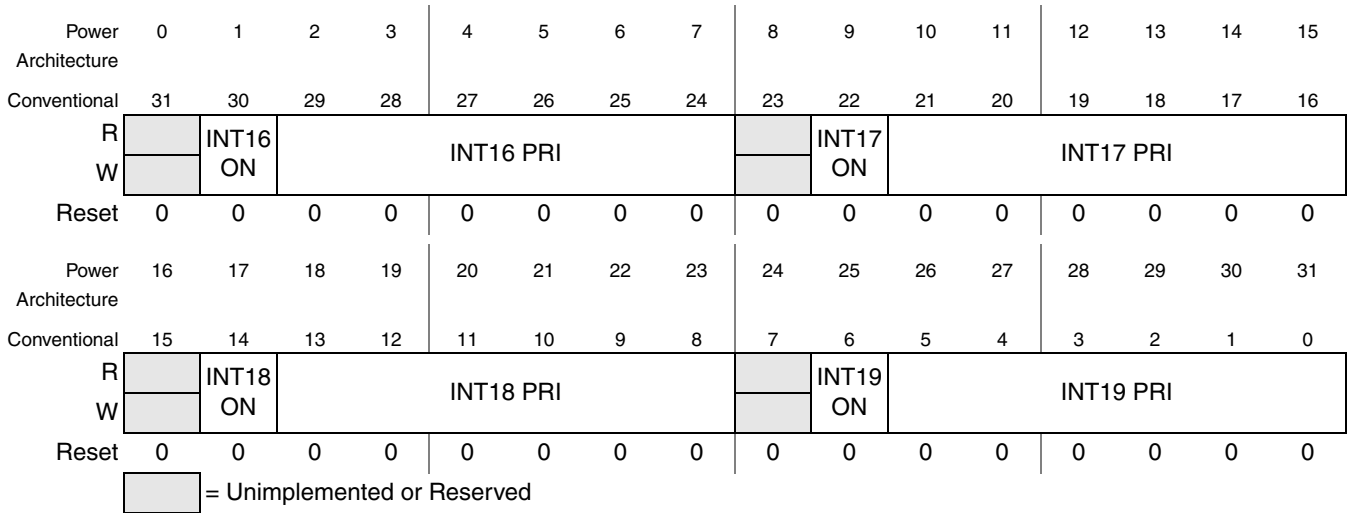


Figure 6-10. INTPRI4 Interrupt Priority Register

Table 6-13. INTPRI4 Interrupt Priority Register Field Descriptions

Field	Description
INT16 ON - INT19 ON	Same as INT0 ON, but for interrupt 16 - 19
INT16 PRI - INT19 PRI	Same as INT0 PRI, but for interrupt 16 - 19

6.2.4.3.6 INTPRI5 Interrupt Priority Register

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		INT20 ON	INT20 PRI							INT21 ON	INT21 PRI					
W		INT20 ON	INT20 PRI							INT21 ON	INT21 PRI					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		INT22 ON	INT22 PRI							INT23 ON	INT23 PRI					
W		INT22 ON	INT22 PRI							INT23 ON	INT23 PRI					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-11. INTPRI5 Interrupt Priority Register

Table 6-14. INTPRI5 Interrupt Priority Register Field Descriptions

Field	Description
INT20 ON - INT23 ON	Same as INT0 ON, but for interrupt 20 - 23
INT20 PRI - INT23 PRI	Same as INT0 PRI, but for interrupt 20 - 23

6.2.4.3.7 INTPRI6 Interrupt Priority Register

Offset 0x28Access: User read/write

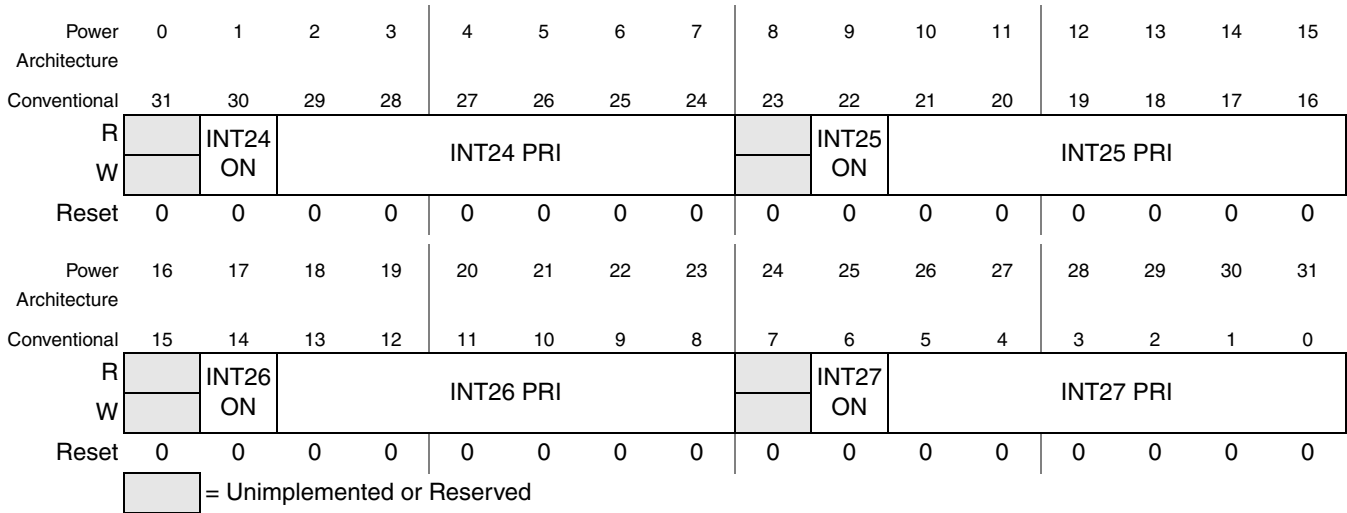


Figure 6-12. INTPRI6 Interrupt Priority Register

Table 6-15. INTPRI6 Interrupt Priority Register Field Descriptions

Field	Description
INT24 ON - INT27 ON	Same as INT0 ON, but for interrupt 24 - 27
INT24 PRI - INT27 PRI	Same as INT0 PRI, but for interrupt 24 - 27

6.2.4.3.8 INTPRI7 Interrupt Priority Register

Offset 0x2C Access: User read/write

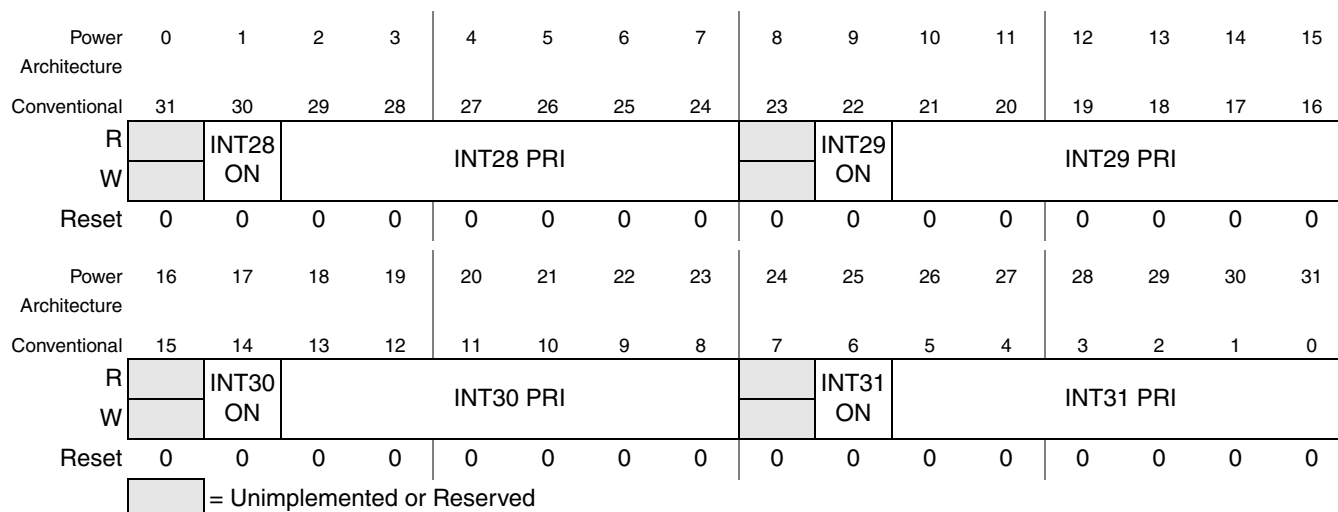


Figure 6-13. INTPRI7 Interrupt Priority Register

Table 6-16. INTPRI7 Interrupt Priority Register Field Descriptions

Field	Description
INT28 ON - INT31 ON	Same as INT0 ON, but for interrupt 28 - 31
INT28 PRI - INT31 PRI	Same as INT0 PRI, but for interrupt 28 - 31

6.2.4.4 Instruction SRAM Indirect Access

6.2.4.4.1 Instruction SRAM Address Register

Offset 0x30Access: User read/write

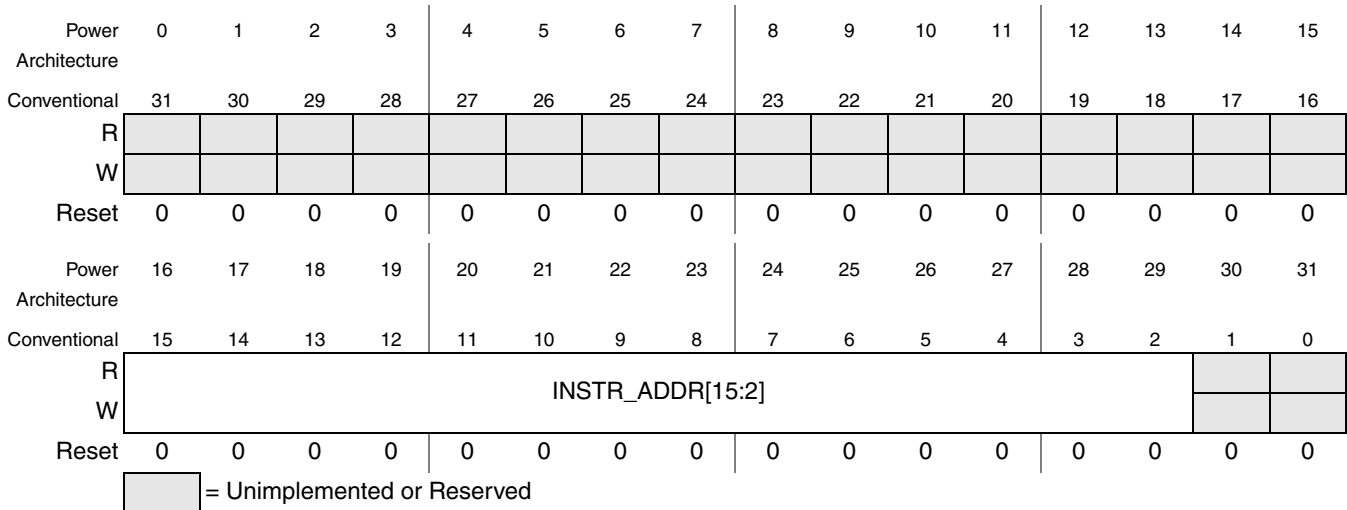


Figure 6-14. Instruction SRAM Address Register

Table 6-17. Instruction SRAM Address Register Field Descriptions

Field	Description
INSTR_ADDR	Address into instruction SRAM. Used for indirect access to the instruction SRAM.

6.2.4.4.2 Instruction SRAM Data Register

Offset 0x34 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INSTR_DATA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INSTR_DATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-15. Instruction SRAM Data Register

Table 6-18. Instruction SRAM Data Register Field Descriptions

Field	Description
INSTR_DATA	Reading INSTR_DATA returns content of instruction SRAM at address INSTR_ADDR. Writing INSTR_DATA writes the data into instruction SRAM at address INSTR_ADDR. Note: Both operation are only valid when the instruction accesses to the instruction SRAM are disabled. IRAM_USER_ON and IRAM_SUP_ON are set to zero.

6.2.4.5 Inter Processor Communication FIFOs

6.2.4.5.1 FIFO1 Write Data Register

Offset 0x3CAccess: User write only

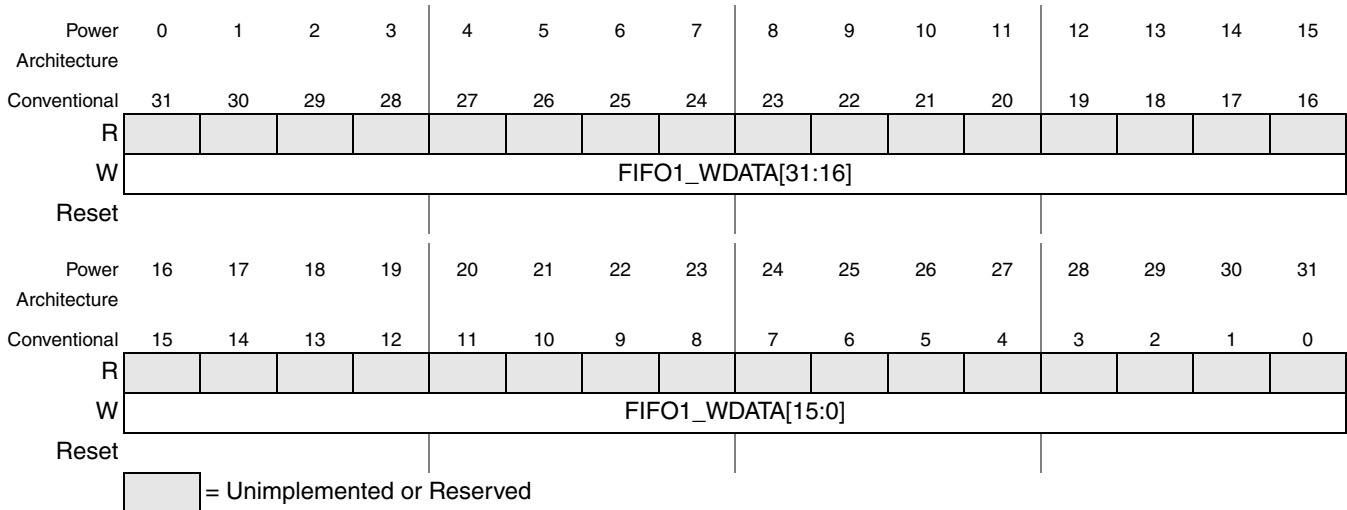


Figure 6-16. FIFO1 Write Data Register

Table 6-19. FIFO1 Write Data Register Field Descriptions

Field	Description
FIFO1_WDATA	FIFO 1 write data Note: Only 32 bit write accesses are allowed.

6.2.4.5.2 FIFO1 Read Data Register

Offset 0x40Access: User read only

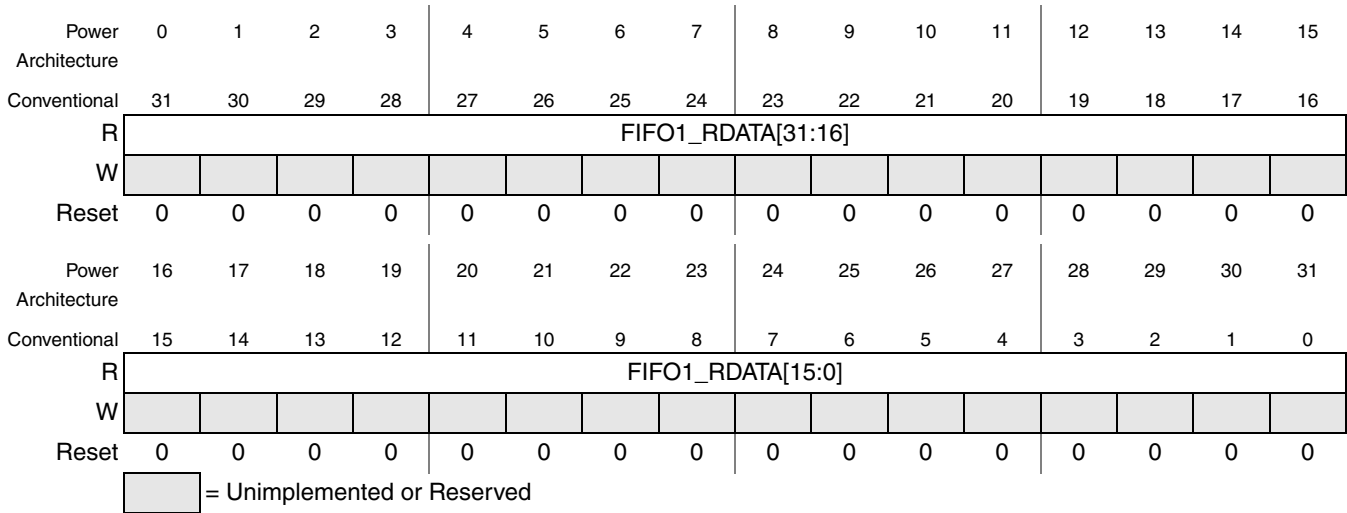


Figure 6-17. FIFO1 Read Data Register

Table 6-20. FIFO1 Read Data Register Field Descriptions

Field	Description
FIFO1_RDATA	FIFO 1 read data Note: Only 32 bit read accesses are allowed.

6.2.4.5.3 FIFO2 Write Data Register

Offset 0x44Access: User write only

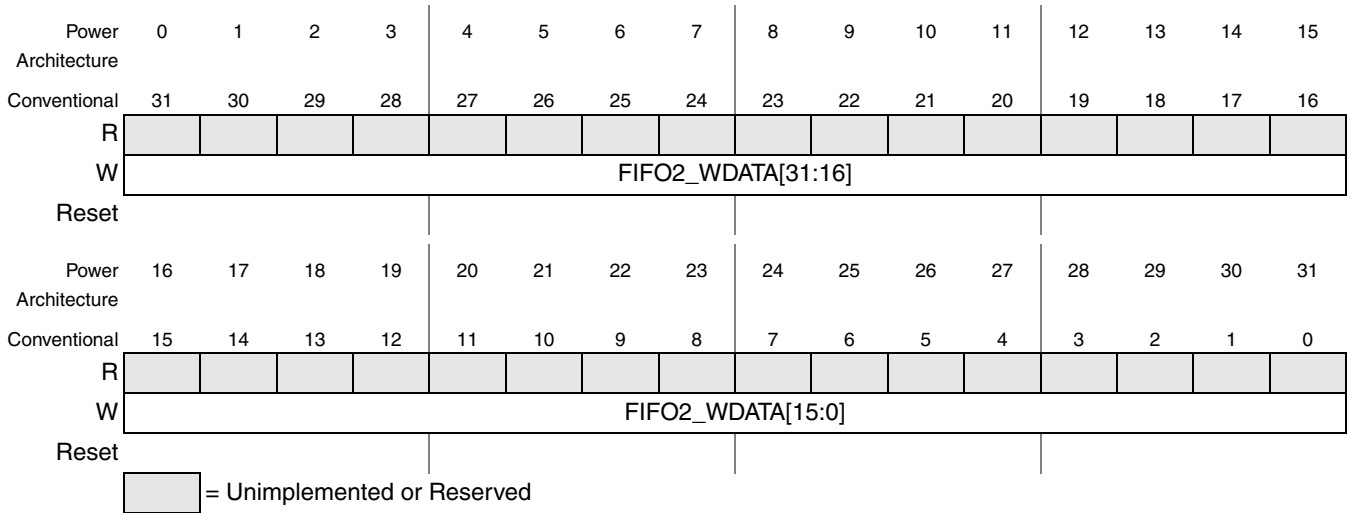


Figure 6-18. FIFO2 Write Data Register

Table 6-21. FIFO2 Write Data Register Field Descriptions

Field	Description
FIFO2_WDATA	FIFO 2 write data Note: Only 32 bit write accesses are allowed.

6.2.4.5.4 FIFO2 Read Data Register

Offset 0x48 Access: User read only

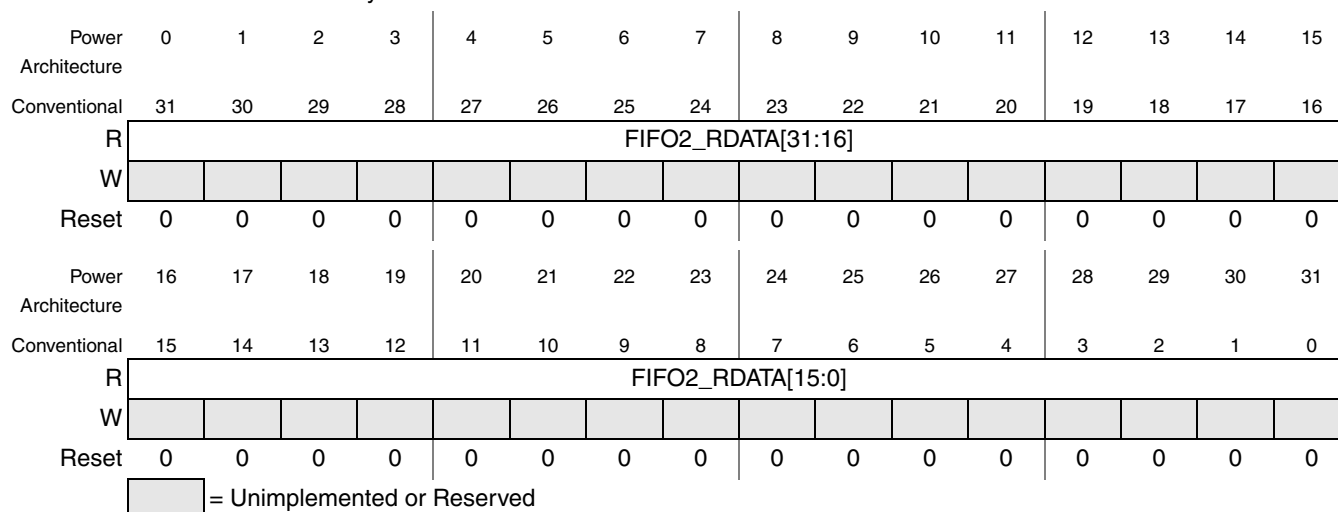


Figure 6-19. FIFO2 Read Data Register

Table 6-22. FIFO2 Read Data Register Field Descriptions

Field	Description
FIFO2_RDATA	FIFO 2 read data Note: Only 32 bit read accesses are allowed.

6.2.4.5.5 FIFO Level Register

Offset 0x4C Access: User read only

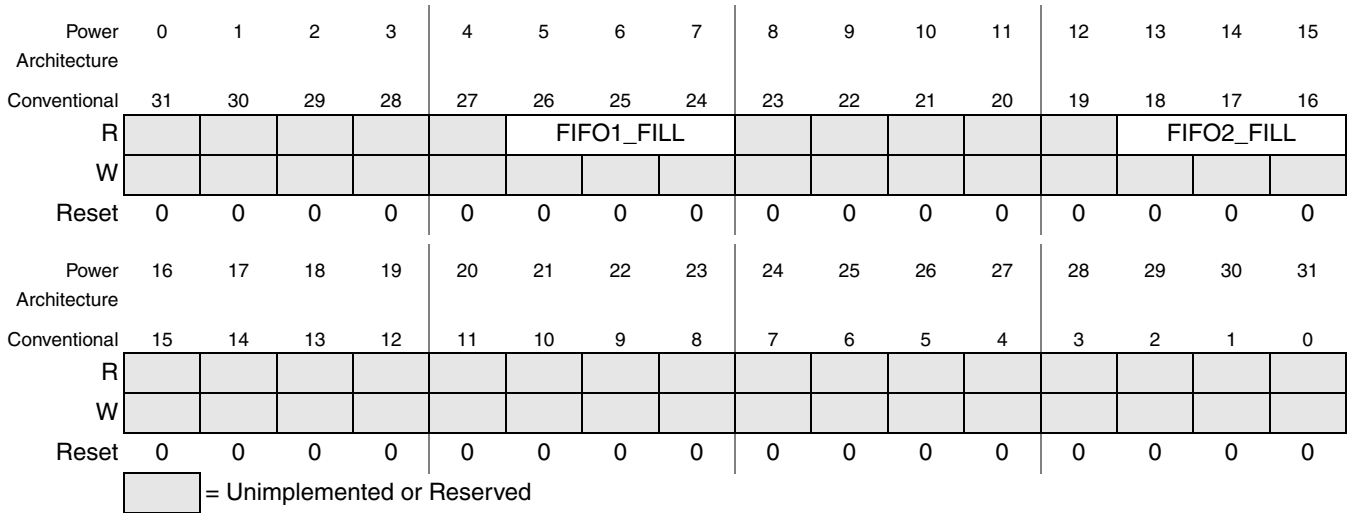


Figure 6-20. FIFO Level Register

Table 6-23. FIFO2 Level Register Field Descriptions

Field	Description
FIFO1_FILL	FIFO 1 level (32-bit words)
FIFO2_FILL	FIFO 2 level (32-bit words)

6.2.4.6 SNE Interrupt Enable Register

Offset 0x50 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SNE SOFTINT[7:0]								SF1EE	SF1NEE	SF1NFE	SF1UE	SF1OE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SF2EE	SF2NEE	SF2NFE	SF2UE	SF2OE											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-21. SNE Interrupt Enable Register

Table 6-24. SNE Interrupt Enable Register Field Descriptions

Field	Description
SNE SOFTINT[7:0]	Any software interrupt can be routed to the SNE processor by setting the corresponding SNE SOFTINT bit.
SF1EE	SNE FIFO1 Empty Enable bit 0: SNE processor doesn't recognize a FIFO1 EMPTY interrupt. 1: FIFO1 EMPTY interrupt is routed to the SNE processor.
SF1NEE	SNE FIFO1 Not Empty Enable bit 0: SNE processor doesn't recognize a FIFO1 NOT EMPTY interrupt. 1: FIFO1 NOT EMPTY interrupt is routed to the SNE processor.
SF1NFE	SNE FIFO1 Not Full Enable bit 0: SNE processor doesn't recognize a FIFO1 NOT FULL interrupt. 1: FIFO1 NOT FULL interrupt is routed to the SNE processor.
SF1UE	SNE FIFO1 Underrun Enable bit 0: SNE processor doesn't recognize a FIFO1 underrun interrupt. 1: FIFO1 underrun interrupt is routed to the SNE processor.
SF1OE	SNE FIFO1 Overrun Enable bit 0: SNE processor doesn't recognize a FIFO1 overrun interrupt. 1: FIFO1 overrun interrupt is routed to the SNE processor.
SF2EE	SNE FIFO2 Empty Enable bit 0: SNE processor doesn't recognize a FIFO2 EMPTY interrupt. 1: FIFO2 EMPTY interrupt is routed to the SNE processor.
SF2NEE	SNE FIFO2 Not Empty Enable bit 0: SNE processor doesn't recognize a FIFO2 NOT EMPTY interrupt. 1: FIFO2 NOT EMPTY interrupt is routed to the SNE processor.
SF2NFE	SNE FIFO2 Not Full Enable bit 0: SNE processor doesn't recognize a FIFO2 NOT FULL interrupt. 1: FIFO2 NOT FULL interrupt is routed to the SNE processor.

Offset 0x50Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SNE SOFTINT[7:0]								SF1EE	SF1NEE	SF1NFE	SF1UE	SF1OE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SF2EE	SF2NEE	SF2NFE	SF2UE	SF2OE											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 6-21. SNE Interrupt Enable Register

Table 6-24. SNE Interrupt Enable Register Field Descriptions (continued)

SF2UE	SNE FIFO2 Underrun Enable bit 0: SNE processor doesn't recognize a FIFO2 underrun interrupt. 1: FIFO2 underrun interrupt is routed to the SNE processor.
SF2OE	SNE FIFO2 Overrun Enable bit 0: SNE processor doesn't recognize a FIFO2 overrun interrupt. 1: FIFO2 overrun interrupt is routed to the SNE processor.

6.2.4.7 Power Architecture Interrupt Enable Register

Offset 0x54 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PPC SOFTINT[7:0]								PF1EE	PF1NEE	PF1NFE	PF1UE	PF1OE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PF2EE	PF2NEE	PF2NFE	PF2UE	PF2OE											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 6-22. Power Architecture Interrupt Enable Register

Table 6-25. Power Architecture Interrupt Enable Register Field Descriptions

Field	Description
PPC SOFTINT[7:0]	Any software interrupt can be routed to the Power Architecture processor by setting the corresponding PPC SOFTINT bit.
PF1EE	Power Architecture FIFO1 Empty Enable bit 0: Power Architecture processor doesn't recognize a FIFO1 EMPTY interrupt. 1: FIFO1 EMPTY interrupt is routed to the Power Architecture processor.
PF1NEE	Power Architecture FIFO1 Not Empty Enable bit 0: Power Architecture processor doesn't recognize a FIFO1 NOT EMPTY interrupt. 1: FIFO1 NOT EMPTY interrupt is routed to the Power Architecture processor.
PF1NFE	Power Architecture FIFO1 Not Full Enable bit 0: Power Architecture processor doesn't recognize a FIFO1 NOT FULL interrupt. 1: FIFO1 NOT FULL interrupt is routed to the Power Architecture processor.
PF1UE	Power Architecture FIFO1 Underrun Enable bit 0: Power Architecture processor doesn't recognize a FIFO1 underrun interrupt. 1: FIFO1 underrun interrupt is routed to the Power Architecture processor.
PF1OE	Power Architecture FIFO1 Overrun Enable bit 0: Power Architecture processor doesn't recognize a FIFO1 overrun interrupt. 1: FIFO1 overrun interrupt is routed to the Power Architecture processor.
PF2EE	Power Architecture FIFO2 Empty Enable bit 0: Power Architecture processor doesn't recognize a FIFO2 EMPTY interrupt. 1: FIFO2 EMPTY interrupt is routed to the Power Architecture processor.
PF2NEE	Power Architecture FIFO2 Not Empty Enable bit 0: Power Architecture processor doesn't recognize a FIFO2 NOT EMPTY interrupt. 1: FIFO2 NOT EMPTY interrupt is routed to the Power Architecture processor.
PF2NFE	Power Architecture FIFO2 Not Full Enable bit 0: Power Architecture processor doesn't recognize a FIFO2 NOT FULL interrupt. 1: FIFO2 NOT FULL interrupt is routed to the Power Architecture processor.

Offset 0x54 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PPC SOFTINT[7:0]								PF1EE	PF1NEE	PF1NFE	PF1UE	PF1OE			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PF2EE	PF2NEE	PF2NFE	PF2UE	PF2OE											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 6-22. Power Architecture Interrupt Enable Register**Table 6-25. Power Architecture Interrupt Enable Register Field Descriptions (continued)**

Field	Description
PF2UE	Power Architecture FIFO2 Underrun Enable bit 0: Power Architecture processor doesn't recognize a FIFO2 underrun interrupt. 1: FIFO2 underrun interrupt is routed to the Power Architecture processor.
PF2OE	Power Architecture FIFO2 Overrun Enable bit 0: Power Architecture processor doesn't recognize a FIFO2 overrun interrupt. 1: FIFO2 overrun interrupt is routed to the Power Architecture processor.

NOTE

The purpose of SNE and Power Architecture Interrupt Enable Registers are to control which active interrupts are sent to the SNE or/and Power Architecture processor. If any bit in this register is set and the same bit in [Figure 6-24](#) is set, an interrupt is made pending to the SNE or/and Power Architecture processor.

6.2.4.8 Interrupt Clear/Set Register

Offset 0x058 Access: User write

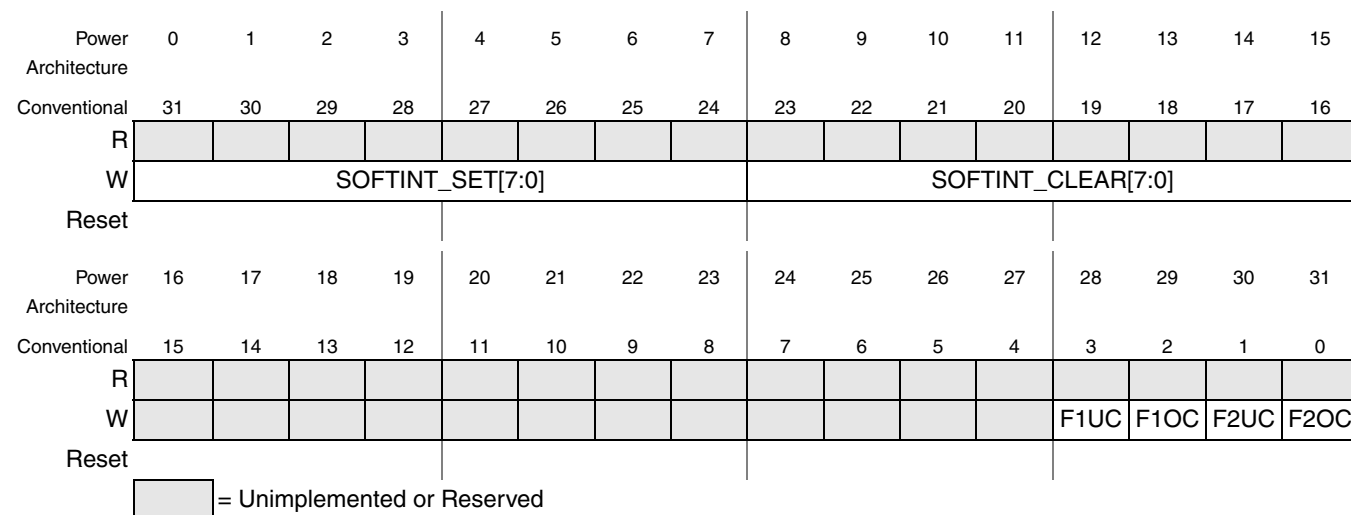


Figure 6-23. Interrupt Clear/Set Register

Table 6-26. Interrupt Clear/Set Register Field Descriptions

Field	Description
SOFTINT_SET	Writing a 1 to any of these bits sets the corresponding bit of the softint field in the interrupt pending register
SOFTINT_CLEAR	Writing a 1 to any of these bits clears the corresponding bit of the softint field in the interrupt pending register.
F1UC	FIFO1 Underrun interrupt Clear bit Writing a 1 to this bit clears the F1UP bit in the interrupt pending register.
F1OC	FIFO1 Overrun interrupt Clear bit Writing a 1 to this bit clears the F1OP bit in the interrupt pending register.
F2UC	FIFO2 Underrun interrupt Clear bit Writing a 1 to this bit clears the F2UP bit in the interrupt pending register.
F2OC	FIFO2 Overrun interrupt Clear bit Writing a 1 to this bit clears the F2OP bit in the interrupt pending register.

NOTE

The FIFO underflow and overflow interrupts are sticky interrupts. They are cleared by writing a 1 to the corresponding bit in the IntClearSet (Section 6.2.4.8, “Interrupt Clear/Set Register”) register. The FIFO data interrupts (empty, not empty, not full), are only influenced by the current FIFO filling. They are not sticky and change from set to clear and clear to set by reading or writing one or more words to/from the corresponding FIFO.

6.2.4.9 Interrupt Pending Register

Offset 0x05C Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SOFTINT[7:0]								F1EP	F1NEP	F1NFP	F1UP	F1OP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	F2EP	F2NEP	F2NFP	F2UP	F2OP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 6-24. Interrupt Pending Register

Table 6-27. Interrupt Pending Register Field Descriptions ¹

Field	Description of interrupts
SOFTINT[7:0]	Software interrupt. There are eight software interrupts. Any software interrupt is made pending when 1 is written to the corresponding bit of field SOFTINT_SET in the IntClearSet register. The software interrupt is cleared when a 1 is written to the corresponding bit of field SOFTINT_CLEAR in the IntClearSet register. Refer to Figure 6-23
F1EP	FIFO1 Empty interrupt Pending bit. Set when FIFO1 is empty. Cleared when FIFO 1 is not empty.
F1NEP	FIFO1 Not Empty interrupt Pending bit. Set when FIFO1 not empty, Cleared when FIFO 1 is empty.
F1NFP	FIFO1 Not Full interrupt Pending bit. Set when FIFO1 is not full. Cleared when FIFO 1 is full.
F1UP	FIFO1 Underrun interrupt Pending bit. Set when FIFO1 experiences a FIFO underflow. Cleared when 1 is written to bit F1UC in the IntClearSet register. Refer to Figure 6-23
F1OP	FIFO1 Overrun interrupt Pending bit. Set when FIFO1 experiences a FIFO overflow. Cleared when 1 is written to bit F1OC in the IntClearSet register. Refer to Figure 6-23
F2EP	FIFO2 Empty interrupt Pending bit. Set when FIFO2 is empty. Cleared when FIFO 2 is not empty.
F2NEP	FIFO2 Not Empty interrupt Pending bit. Set when FIFO 2 not empty. Cleared when FIFO 2 is empty.
F2NFP	FIFO2 Not Full interrupt Pending bit. Set when FIFO 2 is not full. Cleared when FIFO 2 is full.

Offset 0x05C Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SOFTINT[7:0]								F1EP	F1NEP	F1NFP	F1UP	F1OP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	F2EP	F2NEP	F2NFP	F2UP	F2OP											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 6-24. Interrupt Pending Register

Table 6-27. Interrupt Pending Register Field Descriptions (continued)¹

Field	Description of interrupts
F2UP	FIFO2 Underrun interrupt Pending bit Set when FIFO2 experiences a FIFO underrun. Cleared when 1 is written to bit F2UC in the IntClearSet register. Refer to Figure 6-23
F2OP	FIFO2 Overrun interrupt Pending bit. Set when FIFO2 experiences a FIFO overflow. Cleared when 1 is written to bit F2OC in the IntClearSet register. Refer to Figure 6-23

¹ As this is the interrupt pending register, all fields herein are interrupt conditions. These conditions control interrupts that can be sent to Power Architecture and/or SNE core when the corresponding bit is set in the enable register. See [Figure 6-21](#) for the SNE and [Figure 6-22](#) for the Power Architecture.

6.2.4.10 Instruction Cache Indirect Access

6.2.4.10.1 Instruction Cache Address Register

Offset 0x60Access: User read/write

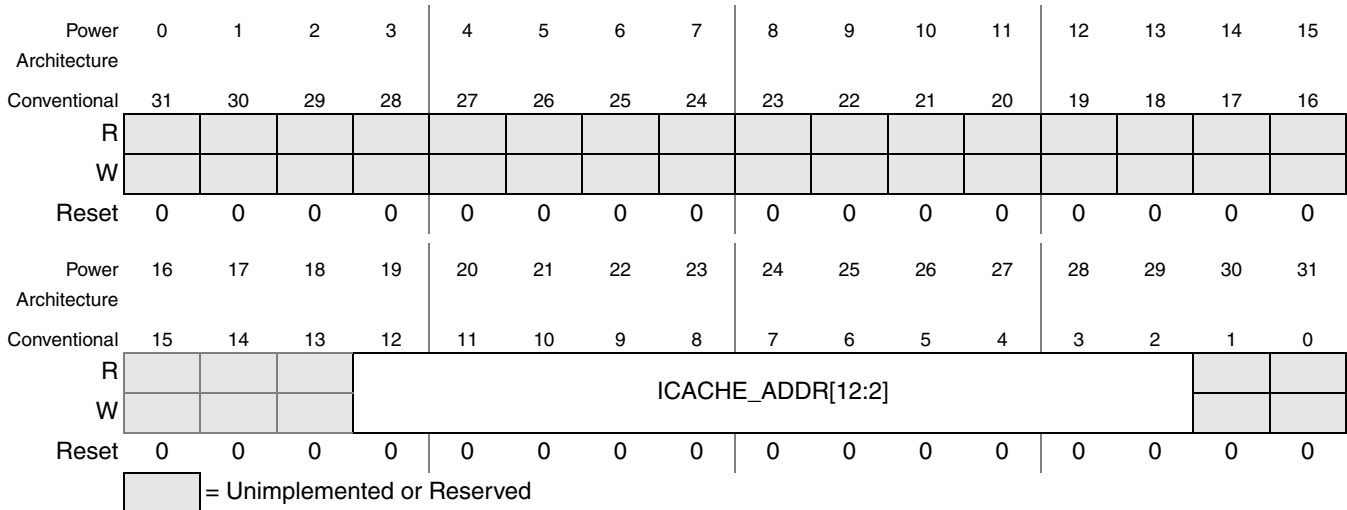


Figure 6-25. Instruction Cache Address Register

Table 6-28. Instruction Cache Address Register Field Descriptions

Field	Description
ICACHE_ADDR	ICACHE_ADDR[12:2]: Address into instruction cache data array ICACHE_ADDR[12:5]: Address into instruction tag data array. (one tag line for every 8 data lines)

6.2.4.10.2 Instruction Cache Data Register

Offset 0x64 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ICACHE_DATA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ICACHE_DATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-26. Instruction Cache Data Register

Table 6-29. Instruction Cache Data Register Field Descriptions

Field	Description
ICACHE_DATA	Reading ICACHE_DATA returns content of instruction cache at address ICACHE_ADDR Writing ICACHE_DATA writes the data into instruction cache at address ICACHE_ADDR Note: Both operation are only valid when the instruction cache is disabled. CACHE_ON is set to zero.

6.2.4.10.3 Instruction Cache Tag Register

Offset 0x68Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													V	S	Addr[31:29]	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Addr[28:13]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-27. Instruction Cache Tag Register

Table 6-30. Instruction Cache Tag Register Field Descriptions

Field	Description
V	Instruction Cache line valid bit 1: Cache line is valid 0 : Cache line is invalid
S	Supervisor bit 1: Cache line contains supervisor memory space data 0: Cache line contains user memory space data
Addr[31:13]	Address tag Cache line contains data corresponding to these memory system MSB address bits

NOTE

- Reading INSTR_CACHE_TAG returns ICACHE_TAG[INSTR_CACHE_ADDR] if the ICACHE is disabled.
- Writing INSTR_CACHE_TAG writes the data into ICACHE_TAG[INSTR_CACHE_ADDR] if the ICACHE is disabled.

6.3 Functional Description

The AXE is made up of the SNE processor, the memory system, and the JTAG tap controller. The AXE memory system contains the instruction cache, an instruction SRAM, the DMA, FIFOs to communicate between the SNE and PPC processor, soft interrupt logic, and the AXE interrupt controller.

6.3.1 AXE Reset

The SNE processor reset is not the power-on reset, but controlled by bit PROC_RST in the instruction and cache attributes register (Table 6-8). When this bit is 1, the SNE is in reset. When this bit is 0, the reset is released. Power-on reset value of the bit is 1.

6.3.2 AXE System

6.3.2.1 AXE System Overview

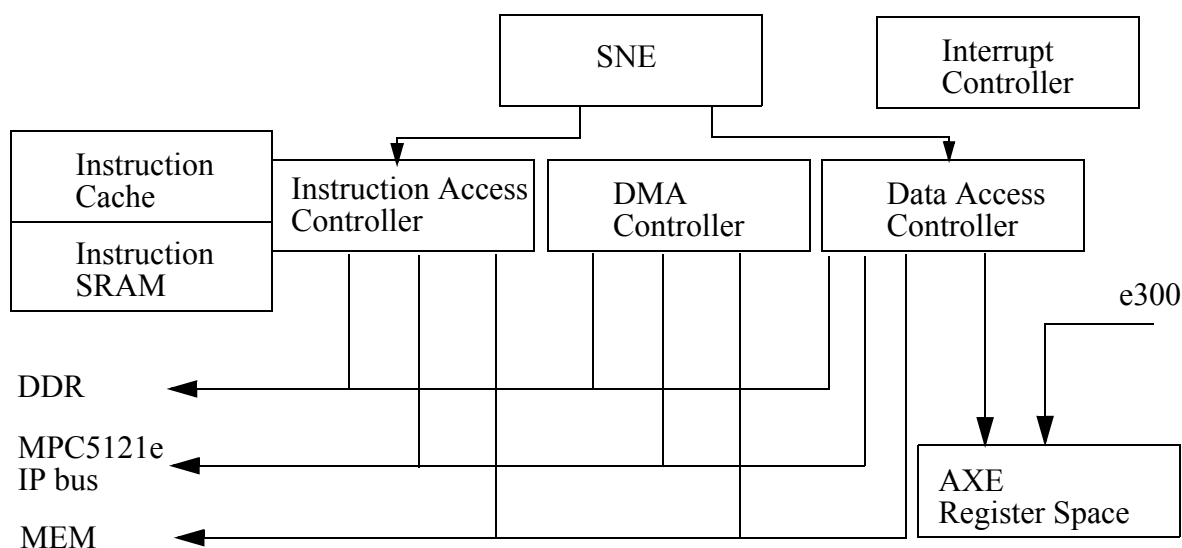


Figure 6-28. AXE Memory System for 5121e

A block diagram of the AXE system is given in Figure 6-28. An overview appears next.

The memory system is connected to the instruction bus and data bus coming from the SNE core. An instruction access controller takes care of the requests coming in over the instruction bus. The data access controller takes care of the requests coming in over the data bus.

Three busses are connected to the memory system. This allows SNE accesses to the on-chip SRAM (MEM), off-chip DRAM and the IP bus.

The AXE system also includes an interrupt controller. It generates the interrupt and their priority to the SNE core.

Through the data access controller, the SNE has a direct connection to the AXE register space. This register space is also accessible by the e300.

All registers needed to program the memory access controllers, the interrupt controller, and the DMA are mapped into this space.

There are some special peripherals in the AXE register space that specifically address the issues of the dual-core system and the inter processor communications.

- There are two FIFOs. Each FIFO is 4x32 bits deep. They can be used for communication between the Power Architecture and the SNE processor. Each FIFO has the possibility to generate full/empty/not empty interrupts to the Power Architecture or the SNE processor.
- There are eight soft-interrupts. Soft-interrupts are interrupts that can be set or reset by writing specific bits in a control register. The soft-interrupts can be set and reset by the Power Architecture or the SNE processor. They can be sent to the Power Architecture or the SNE processor. In this way, they can provide interprocessor communication.

6.3.2.2 Instruction Access Controller

The instruction access controller has five memory systems connected. Two of these are first-priority, meaning they are tried first. Three are second priority, so they are accessed when the first two fail to respond to the request.

- First priority memory systems
 - 2-Kbyte instruction SRAM
 - 8-Kbyte instruction cache
- Second priority memory systems
 - DDR
 - IP bus
 - On-chip SRAM memory

Operation of the memories is now given in detail.

6.3.2.2.1 Instruction SRAM

The instruction SRAM is always mapped to the first 2 Kbytes of the instruction space. It can be enabled or disabled separately for supervisor and user accesses. Enabling/disabling is done by control bits `IRAM_USER_ON` and `IRAM_SUP_ON` in the Instruction Cache and SRAM Attributes Register (Table 6-8). Bit `IRAM_USER_ON` determines if the instruction SRAM is accessible in user mode (1=accessible, 0=inaccessible). Bit `IRAM_SUP_ON` determines if the instruction SRAM is accessible in supervisor mode (1=accessible, 0=inaccessible).

In the instruction space, it's only possible to read from the instruction SRAM. To make it possible to write the SRAM too, and to inspect its contents, there is an indirect access possible to the instruction SRAM. To read or write from the instruction SRAM using this indirect access, proceed as follows:

- Disable all instruction accesses to the SRAM by writing `IRAM_USER_ON` and `IRAM_SUP_ON` to 0.
- Write the address where you want to read or write to the SRAM to the Instruction SRAM Address Register (Table 6-17)
- Read or write the content of the memory from/to Instruction SRAM Data Register (Table 6-18)

6.3.2.3 Instruction Cache

The instruction cache is a one-way set-associative cache, with 32-byte cache lines. The cache size is 8-Kbyte.

All instruction accesses that miss in the instruction SRAM are mapped to the cache first. If they hit in the cache, zero-wait cycle access is done. If they miss in the cache, they are forwarded to the local memory and the bus.

Cache operation can be controlled by two bits in the Instruction Cache and SRAM Attributes Register.

- Bit `CACHE_ON` determines if the cache is enable or not
- Bit `CACHE_CLEAR` is a write-only bit. Writing 1 to this bit flushes the cache. During cache flush, the cache is disabled.

When an access misses in the cache, the instruction access controller loads the complete 32-byte line where this access is part of the local memory or the bus. The word that was needed, is loaded first. In case of a access, all data is loaded with a 32-byte line request.

6.3.2.4 Instruction Accesses Mapped to the On-Chip SRAM

If the address bits [31:20] of an instruction access are equal to the value programmed in SRAM Base Address Register (SRAMBAR), within XLBMEN memory map, this instruction access is mapped to the on-chip SRAM (MEM) if this SRAM is enabled for it. The on-chip SRAM can be enabled separately for supervisor accesses and user accesses. The control bits that enable/disable the local SRAM for instruction accesses are in the Instruction Cache and SRAM Attributes Register.

- If `SRAM2_IUSER_ON` is set, mapping to local SRAM is enabled for mode instruction accesses.
- If `SRAM2_ISUP_ON` is set, mapping to local SRAM is enabled for supervisor instruction accesses.

NOTE

Accesses that hit in the instruction RAM (IRAM) are never forwarded to the on-chip SRAM.

6.3.3 Data Access Controller

The data access controller has two functions:

- It maps the SNE data accesses to the relevant memory system.
- There is a build-in DMA that can do transfers between the DDR or IP bus and the on-chip SRAM.

6.3.3.1 Data Accesses Mapped to the On-Chip SRAM

On-chip SRAM is first-priority memory system for data accesses.

If the address bits [31:20] of a data access are equal to the value programmed in SRAM Base Address Register (SRAMBAR), within XLBMEN register map, this data access is mapped to the on-chip SRAM if this SRAM is enabled for it. The on-chip SRAM can be enabled separately for supervisor accesses and

user accesses. The control bits that enable/disable the on-chip SRAM for instruction accesses are in the Instruction Cache and SRAM Attributes Register.

- If bit SRAM2_DUSER_ON is set, mapping to on-chip SRAM is enabled for user mode data accesses.
- If bit SRAM2_DSUP_ON is set, mapping to on-chip SRAM is enabled for user supervisor data accesses.

6.3.3.2 Data Accesses Mapped to the AXE IP Bus

Data accesses in the first 2 Kbytes of address space are mapped to the AXE IP bus if they do not hit in the on-chip SRAM (MEM) address space.

6.3.3.3 Data Accesses Mapped to the DDR Interface

Data access falling into the DDR memory space, specified by the DDR Local Access Window Base Address Register (DDRLAWBAR) and DDR Local Access Window Attributes Register (DDRLAWAR), which are part of the XLBMEN register map, are mapped to DDR.

6.3.3.4 Data Accesses Mapped to the IP Bus

Data access not hit into the on-chip SRAM, AXE IP Bus or DDR memory space are going to the IP bus.

6.3.4 DMA

There is a DMA capable of moving data between the on-chip SRAM (MEM) and other peripherals. To use the DMA, proceed as follows:

The DMA has three associated registers :

6.3.4.1 DMA_ADDR[31:2] Register (Address 0x0000)

This register contains the first or next address the DMA uses on the DDR or IP bus. It is auto-incremented by the DMA. Do not write this register while the DMA is running, this may result in indeterminate operation of the DMA. It is valid to read this register when the DMA is running, but it changes as it is auto-incremented every time an access is done.

6.3.4.2 DMA_MEM_ADDR[19:2] Register (Address 0x0004)

This register contains the first or next address the DMA uses on the on-chip SRAM (MEM) bus. It is auto-incremented by the DMA. Do not write this register while the DMA is running. It is valid to read this register when the DMA is running, but it changes as it is auto-incremented every time a local bus access is done.

6.3.4.3 DMA attributes Register (Address 0x0008)

This register contains various field. A list is given below:

- **WORD_COUNT[13:0]**. The DMA transfer count in words. Do not write while the DMA is running, or indeterminate operation results. It is valid to read while the DMA is running, but the count is updated every time data is transferred.
- **DMA_DIRECTION**. This bit controls the direction of the transfer. If 1, data is read from the on-chip SRAM (MEM), and written to DDR or IP Bus. If 0, data is read from the bus, and written to the local memory.
- **DMA_ON**. Controls if the DMA is on or not.
- **INT_EN**, **INT_PEND**, **INT_CLEAR**. These bits control the DMA interrupt. At the end of a DMA transfer, a DMA interrupt is generated. At the end of the transfer, **INT_PEND** is set. If **INT_EN** is set, an interrupt is generated. If **INT_EN** is cleared, the interrupt is not forwarded to the SNE processor. Writing the **INT_CLEAR** bit always clears the pending interrupt.

6.3.4.4 Functional Description

The direction must be written (DMA-DIRECTION bit):

- Disable the DMA by writing **DMA_ON** to 0
- Write the DRAM/IP bus address
- Write the SRAM address to **DMA_MEM_ADDR[19:2]**
- Write the desired count to **WORD_COUNT[13:2]**
- Enable or disable the interrupt as desired.
- Start the DMA by writing 1 to **DMA_ON**
- When transfer is complete, **INT_PEND** bit is set and interrupt is made pending (if enabled)

Be aware of the following:

- There is a minimum length of a DMA transfer.
 - If the start address on the bus is aligned to a line boundary (**DMA_ADDR[4:2] == 0**), there are no restrictions.
 - If the start address on the bus is not aligned to a line boundary, the transfer must run to at least the end of the first line.
 - If the transfer size is at least seven longwords, there is no problem.
- The DMA always works with word-aligned addresses. Transferring byte-aligned or halfword-aligned is not possible.
- The DMA always transfers in increments of words. Transferring bytes or halfwords is not possible.
- The DMA always auto-increments the source and destination address.
- The requests on **DMA_ADDR[31:2]** side are always directly mapped to the bus regardless if they hit in the on-chip SRAM space or the AXE IP space.
- The requests on the **DMA_MEM_ADDR[19:2]** side are always directly mapped to the SRAM.
- The DMA attempts to organize transfers on the bus in bursts, but the bursts never cross a line boundary (32 byte boundary).

6.3.5 Interrupt Controller

There are eight registers associated with the interrupt controller: INTPRI0, INTPRI1, INTPRI2, INTPRI3, INTPRI4, INTPRI5, INTPRI6, and INTPRI7.

The interrupt controller services 32 interrupts. Every interrupt has a 6-bit INTXX_PRI[5:0] associated in the control registers and a 1-bit INTXX_ON bit.

The fields for interrupt 12 are INT12_PRI[5:0] and INT12_ON. The field INT12_PRI[5:0] codes for the interrupt vector for interrupt 12. If interrupt 12 is the highest priority, enabled and pending interrupt, interrupt vector INT12_PRI[5:0] is presented to the AXE core on its IRN[5:0] input. At the same time, the AXE core interrupt input IRQ is pulled high. If the bit INT12_ON is 0, interrupt 12 is disregarded. If the bit is 1, the interrupt is processed if pending.

If more than one interrupt is active and enabled at any given time, the interrupt controller arbitrates and presents the highest priority interrupt. This is done by taking the minimum of the INTXX_PRI fields of the pending interrupts and presents this interrupt vector to the core.

Example: Suppose INT12_PRI[5:0] = 6'd23; INT15_PRI[5:0] = 6'd15; INT29_PRI[5:0] = 6'd17, and INTERRUPTS 12, 15, and 29 are enabled and active. The interrupt controller now presents interrupt vector 6'd15 to the core, the vector of interrupt 15, because 6'd15 is the lowest interrupt vector of the three active interrupts.

Table 6-31. List of Interrupts

Interrupt Number	Description
0	PSC FIFO RX0 AXE Request
1	PSC FIFO RX1 AXE Request
2	PSC FIFO RX2 AXE Request
3	PSC FIFO RX3 AXE Request
4	PSC FIFO RX4 AXE Request
5	PSC FIFO RX5 AXE Request
6	PSC FIFO RX6 AXE Request
7	PSC FIFO RX7 AXE Request
8	PSC FIFO RX8 AXE Request
9	PSC FIFO RX9 AXE Request
10	PSC FIFO RX10 AXE Request
11	PSC FIFO RX11 AXE Request
12	PSC FIFO TX0 AXE Request
13	PSC FIFO TX1 AXE Request
14	PSC FIFO TX2 AXE Request
15	PSC FIFO TX3 AXE Request

Table 6-31. List of Interrupts

Interrupt Number	Description
16	PSC FIFO TX4 AXE Request
17	PSC FIFO TX5 AXE Request
18	PSC FIFO TX6 AXE Request
19	PSC FIFO TX7 AXE Request
20	PSC FIFO TX8 AXE Request
21	PSC FIFO TX9 AXE Request
22	PSC FIFO TX10 AXE Request
23	PSC FIFO TX11 AXE Request
24	SPDIF TX DMA Request
25	SPDIF RX DMA Request
26	DMA Interrupt
27	Timer6 and Timer7 interrupts
28	Software interrupts
29	FIFO1 and FIFO2 underflow and overflow interrupts
30	FIFO1 and FIFO2 not full, empty and not empty interrupts
31	Local DMA interrupt

6.3.6 FIFOs for Inter-Processor Communication

There are two FIFOs for inter-processor communication, both resident as an IP peripheral.

The FIFOs operate on longwords, and each FIFO can hold a maximum of four longwords.

Five registers are directly associated with the operation of the FIFOs:

1. FIFO1_WRITE_DATA:¹ word write only register. All data written to this register ends up in FIFO1. Side-effect of writes to this register is that FIFO1 filling increments.
2. FIFO1_READ_DATA:² word read only register. All data read from this register comes from FIFO1. Side-effect of reads to this register is that FIFO1 filling decrements.
3. FIFO2_WRITE_DATA:¹ word write only register. All data written to this register ends up in FIFO2. Side-effect of writes to this register is that FIFO2 filling increments.
4. FIFO2_READ_DATA:² word read only register. All data read from this register comes from FIFO2. Side-effect of reads to this register is that FIFO2 filling decrements.
5. FIFO_FILL: Read-only register. This register contains a 3-bit fill field for FIFO1 and a 3-bit fill field for FIFO2. The fill fields code for the current filling of these FIFO's.

1. If data is written to a full FIFO, the write data is discarded (filling remains at the maximum), and the FIFO's overflow flag is set.

2. If data is read from an empty FIFO, the read data is indeterminate, the FIFO's underflow flag is set, and the filling remains zero.

On top, the FIFO's generate five interrupts for each FIFO:

1. FIFO1_EMPTY, FIFO2_EMPTY: These interrupts are set when the corresponding FIFO is empty. They are cleared after data has been written to the corresponding FIFO.
2. FIFO1_NOT_EMPTY, FIFO2_NOT_EMPTY: These interrupts are set when the corresponding FIFO is not empty. They are cleared after data has been read until the corresponding FIFO is empty.
3. FIFO1_NOT_FULL, FIFO2_NOT_FULL: These interrupts are set when the corresponding FIFO is not full. They are not set if the corresponding FIFO is full.
4. FIFO1_OV, FIFO2_OV: FIFO overflow interrupts. These interrupts are set when an overflow occurs on the corresponding FIFO. Overflow occurs when data is written to a full FIFO. These bits are sticky bits. They are cleared by writing the corresponding bit in register (Table 6-26) IntClearSet.
5. FIFO1_UV, FIFO2_UV: FIFO underflow interrupts. These interrupts are set when an underflow occurs on the corresponding FIFO. Underflow occurs when data is read from an empty FIFO. These bits are sticky bits. They are cleared by writing the corresponding bit in register (Table 6-26) IntClearSet.

6.3.7 Interrupt Enable/Pending and Clear/Set Registers for FIFO1, FIFO2, and Soft Interrupts

These registers affect interrupts SOFTINT[7:0], FIFO1_EMPTY, FIFO2_EMPTY, FIFO1_NOT_EMPTY, FIFO2_NOT_EMPTY, FIFO1_NOT_FULL, FIFO2_NOT_FULL, FIFO1_OV, FIFO2_OV, FIFO1_UV and FIFO2_UV.

6.3.7.1 Setting and Clearing Soft Interrupts

The AXE system supports eight soft interrupts. These are called SOFTINT[7:0]. These soft interrupts can be made pending by writing a 1 to the corresponding set bit of register IntClearSet (Section 6.2.4.8, “Interrupt Clear/Set Register”). Writing a 1 to the corresponding clear bit of this register clears the soft interrupt.

6.3.7.2 Interrupt Enable Registers

There are two interrupt enable registers SNE_INTEN and PPC_INTEN (Section 6.2.4.7, “Power Architecture Interrupt Enable Register”). The register SNE_INTEN controls if FIFO interrupts and soft interrupts are sent to the AXE processor. A 1 in the corresponding interrupt bit means the interrupt is sent to the SNE processor if its pending. The register PPC_INTEN is equal in formatting to SNE_INTEN. Interrupts enabled in this register are sent to the Power Architecture processor.

6.3.7.3 Interrupt Connections

There is one Power Architecture interrupt associated with all soft and FIFO interrupts. All interrupts enabled for the Power Architecture uses the same vector.

There are four AXE interrupts associated with the soft interrupts and the FIFO interrupts. [Table 6-32](#) gives the details.

Table 6-32. AXE Interrupts

Interrupt	SNE Interrupt Number
SOFTINT[7]	28
SOFTINT[6]	28
SOFTINT[5]	28
SOFTINT[4]	28
SOFTINT[3]	28
SOFTINT[2]	28
SOFTINT[1]	28
SOFTINT[0]	28
FIFO1_EMPTY	30
FIFO1_NOT_EMPTY	30
FIFO1_NOT_FULL	30
FIFO1_OV	29
FIFO1_UV	29
FIFO2_EMPTY	30
FIFO2_NOT_EMPTY	30
FIFO2_NOT_FULL	30
FIFO2_OV	29
FIFO2_UV	29
DMADONE	31

Chapter 7

Byte Data Link Controller (BDLC)

7.1 Introduction

The BDLC module is a serial communication module which allows the user to send and receive messages across a Society of Automotive Engineers (SAE) J1850 serial communication network. The user's software manages each transmitted or received message on a byte-by-byte basis, while the BDLC performs all of the network access, arbitration, message framing and error detection duties.

It is recommended that the reader be familiar with the operation and requirements of the SAE J1850 protocol as described in SAE Standard J1850 Class B Data Communications Network Interface document prior to proceeding with this specification.

The BDLC module is designed in a modular structure for use as an IP block. A general working knowledge of the IP bus signals and bus control is assumed in the writing of this document.

Figure 7-1 shows the organization of the BDLC module. The Tx/Rx shadow register function as Buffers provide storage for data received and data to be transmitted onto the J1850 bus. The Protocol Handler is responsible for the encoding and decoding of data bits and special message symbols during transmission and reception. The MUX Interface provides the link between the BDLC digital section and the analog Physical Interface. The wave shaping, driving and digitizing of data is performed by the Physical Interface.

The Physical Interface is not implemented in the BDLC module and must be provided externally.

The main functional blocks of the BDLC module are explained in greater detail in the following sections.

Use of the BDLC module in message networking fully implements the SAE Standard J1850 Class B Data Communication Network Interface specification.

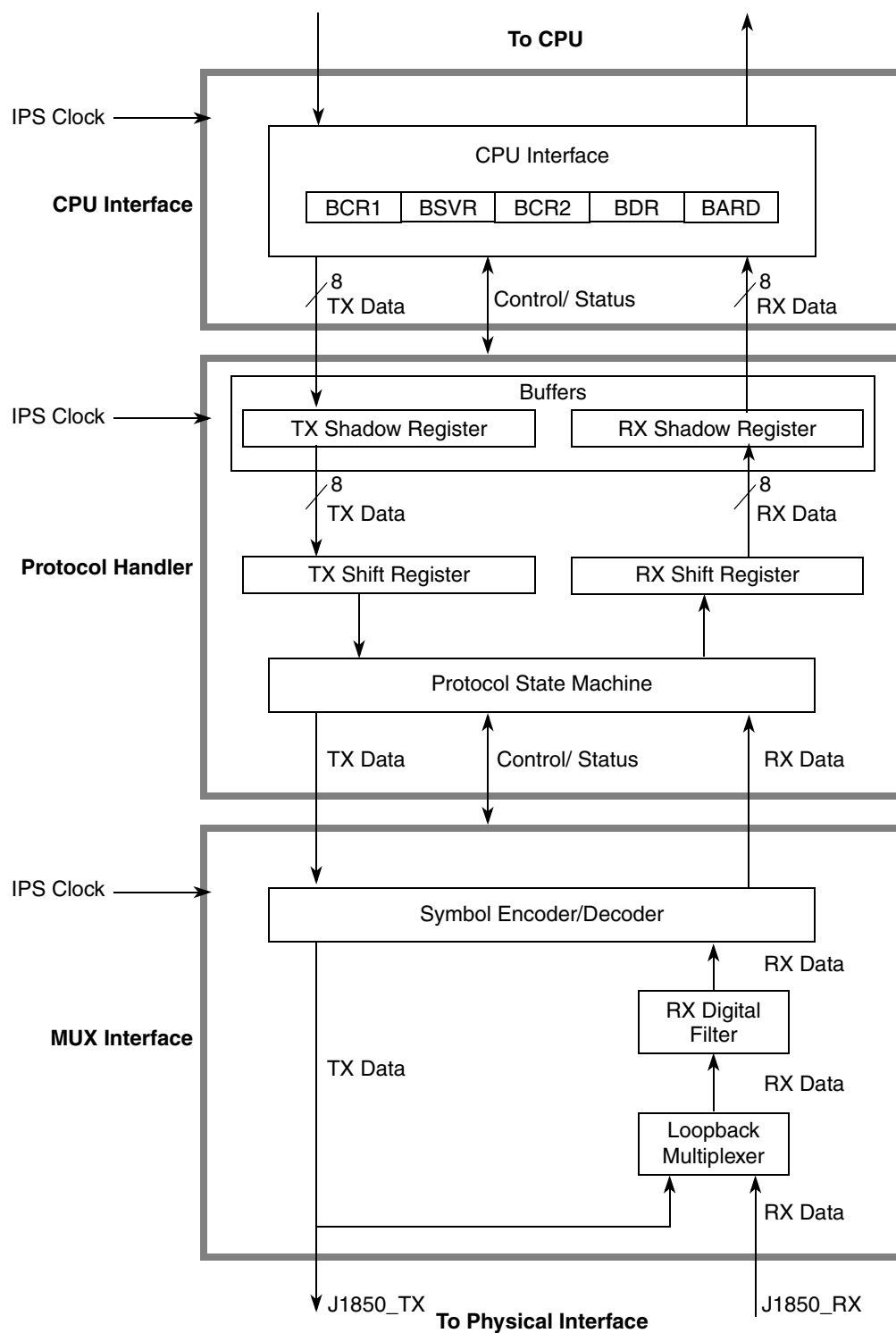


Figure 7-1. BDLC Block Diagram

7.1.1 Features

Features of the BDLC module include the following:

- SAE J1850 Class B Data Communications Network Interface Compatible and ISO Compatible for Low-Speed (≤ 125 Kbps) Serial Data Communications in Automotive Applications
- 10.4 Kbps Variable Pulse Width (VPW) Bit Format
- Digital Noise Filter
- Digital Loopback Mode
- 4X Receive and Transmit Mode, 41.6 Kbps, Supported
- BREAK symbol generation Supported
- Block Mode Receive and Transmit Supported
- Collision Detection
- Hardware Cyclical Redundancy Check (CRC) Generation and Checking
- Dedicated Register for Symbol Timing Adjustments
- In-Frame Response (IFR) Types 0, 1, 2, and 3 Supported
- Polling and CPU Interrupt Generation with Vector Lookup Available

7.2 External Signal Description

The BDLC module has a total of two external pins.

Table 7-1. Signal Properties

Name	Port	Function	I/O	Reset	Pull Up
J1850_TX	Output	The J1850_TX pin serves as the transmit output channel for the BDLC module	O	0	
J1850_RX	Input	The J1850_RX pin serves as the receive input channel for the BDLC module	I		

7.3 Memory Map and Register Definition

7.3.1 Memory Map

The BDLC memory map is shown in [Table 7-2](#).

Table 7-2. BDLC Memory Map

Offset	Register	Access	Reset Value	Section/Page
0x00	BDLC control register 1 (DLCBCR1)	R/W	0xc0	7.3.2.1/7-5
0x01	BDLC state vector register (DLCBSVR)	R	0x00	7.3.2.2/7-6
0x04	BDLC control register 2 (DLCBCR2)	R/W	0x40	7.3.2.3/7-8
0x05	BDLC data register (DLCBDR)	R/W	0x00	7.3.2.5/7-14
0x08	BDLC analog round trip delay register (DLCBARD)	R/W	0x50	7.3.2.6/7-16

Table 7-2. BDLC Memory Map (continued)

Offset	Register	Access	Reset Value	Section/Page
0x09	BDLC rate select register (DLCBRSR)	R/W	0x00	7.3.2.7/7-19
0x0C	BDLC control register (DLCSCR)	R/W	0x00	7.3.2.8/7-20
0x0D	BDLC bus state register (DLCBSTAT)	R/W	0x00	7.3.2.9/7-20

7.3.2 Register Summary

Table 7-3. BDLC Register Summary

Name		7	6	5	4	3	2	1	0
0x00 DLCBCR1	R	IMSG	CLKS	0	0	0	0	IE	WCM
	W								
0x01 DLCBSVR	R	0	0	I3	I2	I1	I0	0	0
	W								
0x04 DLCBCR2	R	SMRST	DLOOP	4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
	W								
0x05 DLCBDR	R	D7	D6	D5	D4	D3	D2	D1	D0
	W								
0x08 DLCBARD	R	0	RXPOL	0	BO4	BO3	BO2	BO1	BO0
	W								
0x09 DLCBRSR	R	R7	R6	R5	R4	R3	R2	R1	R0
	W								
0x0C DLCSCR	R	0	0	0	BDLCE	0	0	0	BREAK
	W								
0x0D DLCBSTAT	R	0	0	0	0	0	tst_divte_t4	tst_crcv_t4	IDLE
	W								

7.3.2.1 BDLC Control Register 1 (DLCBCR1)

This register is used to configure and control the BDLC module.

Read: any time

Write: any time

Offset + 0x00

Access: User read/write

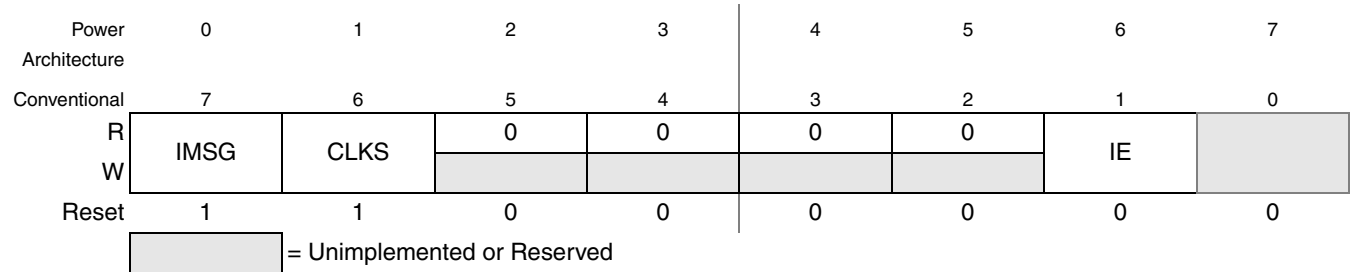


Figure 7-2. BDLC Control Register 1 (DLCBCR1)

Table 7-4. DLCBCR1 Field Descriptions

Field	Description
IMSG	<p>Ignore Message</p> <p>This bit allows the CPU to ignore messages by disabling updates of the BDLC State Vector Register register until a new Start of Frame (SOF) or a BREAK symbol is detected. BDLC module transmitter and receiver operation are unaffected by the state of the IMSG bit. There are two situations in which interrupts are not masked by the IMSG bit: when a wakeup interrupt occurs; and when a receiver error occurs which causes a byte pending transmission to be flushed from the transmit shadow register. See Section 7.3.2.5, BDLC Data Register (DLCBDR) for a description of the conditions which cause a pending transmission to be flushed.</p> <p>0 Enable BDLC State Vector Register Updates. This bit is automatically cleared by the reception of a SOF symbol or a BREAK symbol. It then allows updates of the state vector register to occur.</p> <p>1 Disable BDLC State Vector Register Updates. When set, all BDLC interrupt sources (exceptions are described below) are prevented from updating BDLC State Vector Register status bits. The behavior of which is as described in Section 7.3.2.2, BDLC State Vector Register (DLCBSVR). Setting IMSG does not clear pending interrupt flags. If this bit is set while the BDLC is receiving or transmitting a message, state vector register updates are inhibited for the rest of the message.</p>
CLKS	<p>Clock Select</p> <p>The nominal BDLC operating frequency (mux interface clock frequency - f_{bdlc}) must always be 1.048576 MHz or 1 MHz for J1850 bus communications to take place properly. The CLKS register bit is provided to allow the user to indicate to the BDLC module which frequency (1.048576 MHz or 1 MHz) is used so that each symbol time can be automatically adjusted. The CLKS bit is a write once bit. All writes to this bit are ignored after the first one.</p> <p>0 Integer frequency (1 MHz) is used for f_{bdlc}.</p> <p>1 Binary frequency (1.048576 MHz) is used for f_{bdlc}.</p> <p>Section 7.4.2.10, J1850 VPW Valid/Invalid Bits and Symbols describes the transmitter and receiver VPW symbol timing for integer and binary frequencies.</p>

Offset + 0x00

Access: User read/write

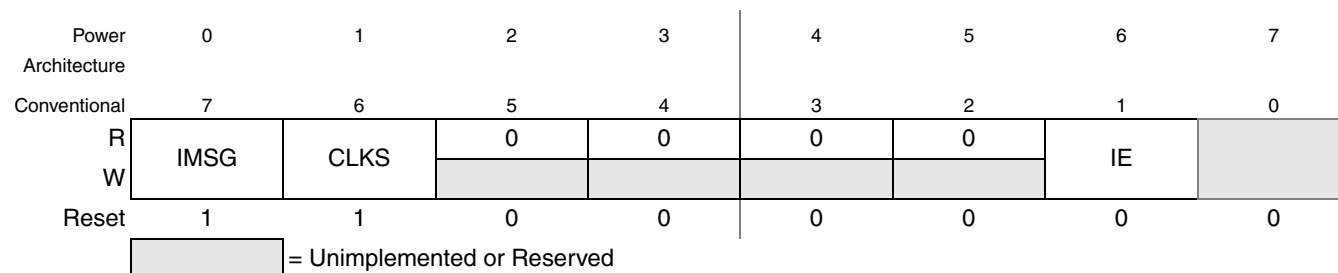


Figure 7-2. BDLC Control Register 1 (DLCBCR1)

Table 7-4. DLCBCR1 Field Descriptions (continued)

Field	Description
IE	<p>Interrupt Enable</p> <p>This bit determines whether the BDLC module generates CPU interrupt requests. Interrupt requests are maintained until all of the interrupt request sources are cleared, by performing the specified actions upon the BDLC module's registers. Interrupts that were pending at the time that this bit is cleared may be lost.</p> <p>0 Disable interrupt requests from BDLC module. 1 Enable interrupt requests from BDLC module.</p> <p>If the programmer does not wish to use the interrupt capability of the BDLC module, the BDLC State Vector Register (BDLC State Vector Register) can be polled periodically by the programmer to determine BDLC module states. Refer to for a description of Section 7.3.2.2, BDLC State Vector Register (DLCBSVR) and how to clear interrupt requests.</p>

7.3.2.2 BDLC State Vector Register (DLCBSVR)

This register substantially decreases the CPU overhead associated with servicing interrupts while under operation of a MUX protocol. It provides an index offset directly related to the BDLC module's current state, which can be used with a user supplied jump table to rapidly enter an interrupt service routine. This eliminates the need for the user to maintain a duplicate state machine in software.

Read: any time

Write: ignored

Encoding interrupt sources in states allows that only one interrupt source has to be managed at a time. After the highest priority interrupt source is dealt with and if another interrupt event of a lower priority has also occurred, the value corresponding to that interrupt source appears in the BDLC State Vector Register. This continues until all BDLC interrupt sources have been managed and all bits in the BDLC state vector register are cleared.

- Symbol invalid or out of range
- CRC error

Cyclical Redundancy Check Byte is used by the receiver(s) of each message to determine if any errors have occurred during the transmission of the message. If the message is not error free, the CRC error status is shown in the BDLC state vector register.

- Loss of arbitration

Loss of arbitration status is entered when a loss of arbitration occurs while the BDLC is transmitting onto the bus.

- Tx data register empty

The Tx data register empty (TDRE) state is used to tell when data has been unloaded from the BDLC Data Register.

- Rx data register full

The Rx data register full (RDRF) state describes when data has been loaded in the BDLC Data Register.

- Received in-frame response (IFR) byte

The BDLC can transmit and receive all four types of in-frame responses. As each byte of an IFR is received, the BDLC State Vector Register indicates this by setting this state.

- Received EOF

When a 280 μ s passive period on the bus is received, it signifies an end-of-frame (EOF). When this occurs, the EOF flag is set.

- No interrupts pending

This interrupt cannot generate an interrupt of the CPU.

Offset + 0x01				Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	I3	I2	I1	I0	0	0
W								
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 7-3. BDLC State Vector Register (DLCBSVR)

Table 7-5. DLCBSVR Field Descriptions

Field	Description
I[3:0]	Interrupt State Vector(with priority from low to high) 0000 No interrupt Pending. 0001 Received EOF 0010 Received IFR byte 0011 Rx data register full 0100 Tx data register empty 0101 Loss of arbitration 0110 CRC error 0111 Symbol invalid or out of range 1000 Reserved, not implemented

7.3.2.3 BDLC Control Register 2 (DLCBCR2)

This register controls transmitter operations of the BDLC module.

Read: any time

Write: any time

7.3.2.4 Transmit Multiple Bytes IFR with CRC (TMIFR1)

This bit requests the BDLC module to transmit the byte in the BDLC Data Register (BDLC data register) as the first byte of a multiple byte IFR with CRC or as a single byte IFR with CRC. Response IFR bytes are subject to J1850 message length maximums.

After the byte in the BDLC Data Register has been loaded into the transmit shift register, the transmit data register empty (TDRE) flag is set in the BDLC State Vector Register register, similar to the main message transmit sequence. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. The programmer should then load the next byte of the IFR into the BDLC Data Register for transmission. When the last byte of the IFR has been loaded into the BDLC Data Register, the programmer should set the transmit end of data (TEOD) bit in the BDLC control register 2. This instructs the BDLC module to transmit a CRC byte once the byte in the BDLC Data Register is transmitted, and then transmit an EOD symbol, indicating the end of the IFR portion of the message frame.

However, if the programmer wishes to transmit a single byte followed by a CRC byte, the programmer should load the byte into the BDLC Data Register and then set the TMIFR1 bit before the EOD symbol has been received. Once the TDRE flag is set and interrupt occurs (if enabled), the programmer should then set the TEOD bit in BDLC Control Register 2. The byte in the BDLC data register is the only byte transmitted before the IFR CRC byte.

Set the TMIFR1 bit before the EOF following the main part of the message frame is received or no IFR transmit attempts are made for the current message. If another node transmits an IFR to this message, the user must set the TMIFR1 bit before the normalization bit is received or no IFR transmit attempts are made for the message. If another node does transmit a successful IFR or a reception error occurs, the TMIFR1 bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

If a transmitter underrun error occurs during transmission (caused by not writing another byte to the BDLC data register following the TDRE flag being set), the BDLC module automatically disables the transmitter after the byte currently in the shifter. Also, two extra 1-bits have been transmitted. The receiver picks this up as an framing error and relay it in the State Vector Register as an invalid symbol error. The TMIFR1 bit is also cleared.

If a loss of arbitration occurs when the BDLC module is transmitting a multiple byte IFR with CRC, the BDLC module switches to the loss of arbitration state and sets the appropriate flag and cease transmission. The TMIFR1 bit is cleared and no attempt is made to retransmit the byte in the BDLC Data Register. If loss of arbitration occurs in the last bit of the IFR byte, two additional one bits (a passive long followed by an active short) are sent out.

NOTE

The extra logic is an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise on the J1850 bus from corrupting a message.

Offset + 0x04				Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	SMRST	DLOOP	4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
W								
Reset	0	1	0	0	0	0	0	0

Figure 7-4. BDLC Control Register 2 (DLCBCR2)
(The register is repeated for reference.)

Table 7-6. DLCBCR2 Field Descriptions

Field	Description
SMRST	<p>State Machine Reset</p> <p>You can use this bit to reset the BDLC state machines to an initial state after the you put the off-chip analog transceiver in loop back mode.</p> <p>0 Clearing SMRST after it has been set causes the generation of a state machine reset. After SMRST is cleared, the BDLC requires the bus to be idle for a minimum of an EOF symbol time before allowing the reception of a message. The BDLC requires the bus to be idle for a minimum of an inter-frame separator symbol (IFS) time before allowing any message to be transmitted.</p> <p>1 Setting SMRST arms the state machine reset generation logic. Setting SMRST does not affect BDLC module behavior in any way.</p>
DLOOP	<p>Digital Loopback Mode</p> <p>This bit determines the input source the digital filter is connected to and can be used to isolate bus fault conditions. If a fault condition has been detected on the bus, this control bit allows the programmer to disconnect the digital filter from input from the receive pin (RXB) and connect it to the transmit output to the pin (TXB). In this configuration, data sent from the transmit buffer should be reflected back into the receive buffer. If no faults exist in the digital block, the fault is in the physical interface block or elsewhere on the J1850 bus.</p> <p>0 No loopback. When cleared, digital filter input is connected to receive pin (J1850_RX) and the transmitter output is connected to the transmit pin (J1850_TX). The BDLC module is taken out of Digital Loopback Mode and can now drive and receive from the J1850 bus normally. After writing DLOOP to zero, the BDLC module requires the bus to be idle for a minimum of an EOF symbol time before allowing a reception of a message. The BDLC module requires the bus to be idle for a minimum of an inter-frame separator symbol time before allowing any message to be transmitted.</p> <p>1 Loopback. When set, digital filter input is connected to the transmitter output. The BDLC module is now in Digital Loopback Mode of operation. The transmit pin (J1850_TX) is driven low and not driven by the transmitter output.</p> <p>Note: The DLOOP bit is a fault condition aid and should never be altered after the BDLC Data Register is loaded for transmission. Changing DLOOP during a transmission may cause corrupted data to be transmitted onto the J1850 network.</p>

Offset + 0x04

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	SMRST	DLOOP	4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
W								
Reset	0	1	0	0	0	0	0	0

Figure 7-4. BDLC Control Register 2 (DLCBCR2)
(The register is repeated for reference.)

Table 7-6. DLCBCR2 Field Descriptions (continued)

Field	Description
4XE	<p>4X Mode Enable</p> <p>This bit determines if the BDLC operates at normal transmit and receive speed (10.4 kbps) or in 4X Mode at 41.6 kbps. This feature is useful for fast download of data into a J1850 node for diagnostic or factory programming of the node. The effect of 4X receive operation on receive symbol timing boundaries is described in Section 7.4.2.10, J1850 VPW Valid/Invalid Bits and Symbols.</p> <p>0 When cleared, the BDLC module transmits and receives at 10.4 kbps. Reception of a BREAK symbol automatically clears this bit and sets the symbol invalid or out of range flag BDLC State Vector Register = 0x1C).</p> <p>1 When set, the BDLC module is put in 4X (41.6 kbps) operation.</p>
NBFS	<p>Normalization Bit Format Select</p> <p>This bit controls the format of the normalization bit (NB). SAE J1850 strongly encourages the use of an active long: 0 for In-Frame Responses containing CRC and active short, 1 for In-Frame Responses without CRC.</p> <p>0 NB that is received or transmitted is a 1 when the response part of an In-Frame Response (IFR) ends with a CRC byte. NB that is received or transmitted is a 0 when the response part of an In-Frame Response (IFR) does not end with a CRC byte.</p> <p>1 NB that is received or transmitted is a 0 when the response part of an In-Frame Response (IFR) ends with a CRC byte. NB that is received or transmitted is a 1 when the response part of an In-Frame Response (IFR) does not end with a CRC byte.</p>
TEOD	<p>Transmit End of Data</p> <p>This bit is set by the programmer to indicate the end of a message being sent by the BDLC. It appends an 8-bit CRC after completing transmission of the current byte in the Tx Shift Register followed by the EOD symbol. If the transmit shadow register (refer to Section 7.4.4.1, Protocol Architecture for a description of the transmit shadow register) is full when TEOD is set, the CRC byte and EOD is transmitted after the current byte in the Tx Shift Register and the byte in the Tx Shadow Register have been transmitted. Once TEOD is set, the transmit data register empty flag (TDRE) in the BDLC state vector register (BDLCSVR) is cleared to allow lower priority interrupts to occur. This bit is also used to end an IFR. Bits TSIFR, TMIFR1, and TMIFR0 determine whether a CRC byte is appended before EOD transmission for IFRs.</p> <p>0 The TEOD bit is automatically cleared after the first CRC bit is sent, or if an error or loss of arbitration is detected on the bus. When TEOD is used to end an IFR transmission, TEOD is cleared when the BDLC receives back a valid EOD symbol, or an error condition or loss of arbitration occurs.</p> <p>1 Transmit EOD symbol</p>
TSIFR	<p>Transmit Single Byte IFR with no CRC (Type 1 or 2)</p> <p>0 The TSIFR bit is automatically cleared after the EOD if one or more IFR bytes has been received or an error is detected on the bus.</p> <p>1 If this bit is set prior to a valid EOD being received with no CRC error and after the EOD symbol has been received, the BDLC module attempts to transmit the appropriate normalization bit followed by the byte in the BDLC Data Register.</p>

Offset + 0x04

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	SMRST	DLOOP	4XE	NBFS	TEOD	TSIFR	TMIFR1	TMIFR0
W								
Reset	0	1	0	0	0	0	0	0

Figure 7-4. BDLC Control Register 2 (DLCBCR2)
(The register is repeated for reference.)

Table 7-6. DLCBCR2 Field Descriptions (continued)

Field	Description
TMIFR1	<p>Transmit Multiple Byte IFR with CRC (Type 3)</p> <p>0 The TMIFR1 bit is automatically cleared once the BDLC module has successfully transmitted the CRC byte and EOD symbol, by the detection of an error on the multiplex bus, a transmitter underrun, or loss of arbitration.</p> <p>1 If this bit is set prior to a valid EOD being received with no CRC error, once the EOD symbol has been received, the BDLC module attempts to transmit the appropriate normalization bit followed by IFR bytes. The programmer should set TEOD after the last IFR byte has been written into BDLC Data Register. After TEOD has been set and the last IFR byte has been transmitted, the CRC byte is transmitted.</p>
TMIFR0	<p>Transmit Multiple Byte IFR with no CRC (Type 3)</p> <p>0 The TMIFR0 bit is automatically cleared once the BDLC module has successfully transmitted the EOD symbol, by the detection of an error on the multiplex bus, a transmitter underrun, or loss of arbitration.</p> <p>1 If this bit is set prior to a valid EOD being received with no CRC error, once the EOD symbol has been received the BDLC module attempts to transmit the appropriate normalization bit followed by IFR bytes. The programmer should set TEOD after the last IFR byte has been written into BDLC Data Register. After TEOD has been set, the last IFR byte to be transmitted is the last byte written into the BDLC Data Register.</p>

The TSIFR, TMIFR1, and TMIFR0 bits control the type of In-Frame Response being sent. The programmer should not set more than one of these control bits to a one at any given time. However, if more than one of these three control bits are set to one, the priority encoding logic forces the internal register bits to a known value as shown in the following table. However, when these bits are read, they are the same as written earlier. For instance, if 011 is written to TSIFR, TMIFR1, and TMIFR0, then internally, they are encoded as 010. However, when these bits are later read back, they are encoded as 011.

The BDLC supports the In-frame Response (IFR) feature of J1850 by setting these bits correctly. The four types of J1850 IFR are shown in [Figure 7-5](#). The purpose of the in-frame response modes is to allow single or multiple nodes to acknowledge receipt of the data by responding to a received message after they have seen the EOD symbol. For VPW modulation, the first bit of the IFR is always passive; therefore, an active normalization bit must be generated by the responder and sent prior to its ID/address byte. When there are multiple responders on the J1850 bus, only one normalization bit is sent which assists all other transmitting nodes to sync their responses.

The TSIFR bit is used to request the BDLC to transmit the byte in the BDLC Data Register as a single byte IFR with no CRC. Typically, the byte transmitted is a unique identifier or address of the transmitting (responding) node.

Set the TSIFR bit before the EOF following the main part of the message frame is received or no IFR transmit attempts are made for the current message. If another node transmits an IFR to this message, set the TSIFR bit before the normalization bit is received or no IFR transmit attempts are made for the message. If another node does transmit a successful IFR or a reception error occurs, the TSIFR bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

If a loss of arbitration occurs when the BDLC module attempts transmission, after the IFR byte winning arbitration completes transmission, the BDLC module again attempts to transmit the byte in the BDLC Data Register (with no normalization bit). The BDLC module continues transmission attempts until an error is detected on the bus, or TEOD is set by the CPU, or the BDLC transmission is successful.

NOTE

Setting the TEOD bit before transmission of the IFR byte directs the BDLC to make only one attempt at transmitting the byte.

If loss of arbitration occurs in the last bit of the IFR byte, two additional 1 bits is not sent out because the BDLC attempts to retransmit the byte in the transmit shift register after the IFR byte winning arbitration completes transmission.

The TMIFR1 bit requests the BDLC module to transmit the byte in the BDLC Data Register (BDLC Data Register) as the first byte of a multiple byte IFR with CRC or as a single byte IFR with CRC. Response IFR bytes are subject to J1850 message length maximums.

After the byte in the BDLC Data Register has been loaded into the transmit shift register, the TDRE flag is set in the BDLC State Vector Register register, similar to the main message transmit sequence. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. The programmer should then load the next byte of the IFR into the BDLC Data Register for transmission. When the last byte of the IFR has been loaded into the BDLC Data Register, the programmer should set the TEOD bit in the BDLC control register 2. This instructs the BDLC module to transmit a CRC byte once the byte in the BDLC Data Register is transmitted, and then transmit an EOD symbol, indicating the end of the IFR portion of the message frame.

However, if you wish to transmit a single byte followed by a CRC byte, load the byte into the BDLC Data Register and then set the TMIFR1 bit before the EOD symbol has been received. Once the TDRE flag is set and interrupt occurs (if enabled), the programmer should then set the TEOD bit in BDLC Control Register 2. This results in the byte in the BDLC Data Register being the only byte transmitted before the IFR CRC byte.

Set the TMIFR1 bit before the EOF following the main part of the message frame is received or no IFR transmit attempts are made for the current message. If another node transmits an IFR to this message, set the TMIFR1 bit before the normalization bit is received or no IFR transmit attempts are made for the message. If another node does transmit a successful IFR or a reception error occurs, the TMIFR1 bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

If a transmitter underrun error occurs during transmission (caused by not writing another byte to the BDLC data register following the TDRE flag being set), the BDLC module automatically disables the transmitter after the byte currently in the shifter. Two extra 1-bits have been transmitted. The receiver picks this up as an framing error and relay it in the state vector register as an invalid symbol error. The TMIFR1 bit is also cleared.

If a loss of arbitration occurs when the BDLC module is transmitting a multiple byte IFR with CRC, the BDLC module goes to the loss of arbitration state, set the appropriate flag and cease transmission. The TMIFR1 bit is cleared and no attempt is made to retransmit the byte in the BDLC Data Register. If loss of arbitration occurs in the last bit of the IFR byte, two additional one bits (a passive long followed by an active short) is sent out.

NOTE

The extra logic is an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise on the J1850 bus from corrupting a message

The TMIFR0 bit is used to request the BDLC module to transmit the byte in the BDLC Data Register as the first byte of a multiple byte IFR without CRC. Response IFR bytes are subject to J1850 message length maximums.

After the byte in the BDLC Data Register has been loaded into the transmit shift register, the TDRE flag is set in the BDLC State Vector Register register, similar to the main message transmit sequence. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. The programmer should then load the next byte of the IFR into the BDLC Data Register for transmission. When the last byte of the IFR has been loaded into the BDLC Data Register, the programmer should set the TEOD bit in the BDLC Control Register 2. This instructs the BDLC to transmit an EOD symbol, indicating the end of the IFR portion of the message frame. The BDLC module does not append a CRC.

However, if the programmer wishes to transmit a single byte, the programmer should load the byte into the BDLC Data Register and then set the TMIFR0 bit before the EOD symbol has been received. Once the TDRE flag is set and interrupt occurs (if enabled), the programmer should then set the TEOD bit in BDLC Control Register 2. This results in the byte in the BDLC Data Register being the only byte transmitted.

Set the TMIFR0 bit before the EOF following the main part of the message frame is received, or no IFR transmit attempts is made for the current message. If another node transmits an IFR to this message, the user must set the TMIFR0 bit before the normalization bit is received or no IFR transmit attempts are made for the message. If another node does transmit a successful IFR or a reception error occurs, the TMIFR0 bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

If a transmitter underrun error occurs during transmission (caused by the programmer not writing another byte to the BDLC Data Register following the TDRE flag being set) the BDLC module automatically disables the transmitter after the byte currently in the shifter plus two extra 1-bits have been transmitted. The receiver picks this up as an framing error and relay it in the State Vector Register as an invalid symbol error. The TMIFR0 bit is also cleared.

If a loss of arbitration occurs when the BDLC module is transmitting a multiple byte IFR without CRC, the BDLC module goes to the loss of arbitration state, set the appropriate flag and cease transmission. The

TMIFR0 bit is cleared and no attempt is made to retransmit the byte in the BDLC Data Register. If loss of arbitration occurs in the last bit of the IFR byte, two additional one bits (a passive long followed by an active short) is sent out.

NOTE

The extra logic is an enhancement to the J1850 protocol which forces a byte boundary condition fault. This is helpful in preventing noise on the J1850 bus from corrupting a message

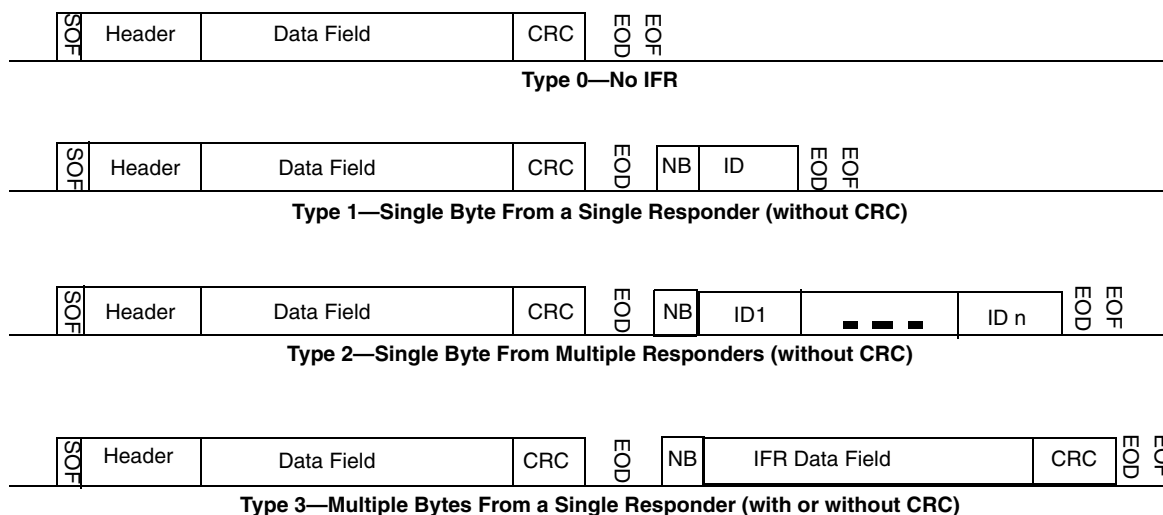


Figure 7-5. Types of In-Frame Response

Table 7-1. Transmit In-Frame Response Control Bit Priority Encoding

WRITE			READ			ACTUAL (Internal Register)		
TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0
0	0	0	0	0	0	0	0	0
1	—	—	1	—	—	1	0	0
0	1	—	0	1	—	0	1	0
0	0	1	0	0	1	0	0	1

7.3.2.5 BDLC Data Register (DLCBDR)

This register is used to pass the data to be transmitted to the J1850 bus from the CPU to the BDLC module. It is also used to pass data received from the J1850 bus to the CPU.

READ: any time

WRITE: any time

While transmitting, each data byte (after the first one) should be written only after a Tx Data Register Empty (TDRE) interrupt has occurred, or the BDLC State Vector Register register has been polled indicating this condition.

Data read from this register is the last data byte received from the J1850 bus. This received data should only be read after a Rx Data Register Full (RDRF) or Received IFR byte (RXIFR) interrupt has occurred or the BDLC State Vector Register register has been polled indicating either of these two conditions.

The BDLC Data Register is double buffered via a transmit shadow register and a receive shadow register. After the byte in the transmit shift register has been transmitted, the byte currently stored in the transmit shadow register is loaded into the transmit shift register. Once the transmit shift register has shifted the first bit out, the TDRE flag is set, and the shadow register is ready to accept the next byte of data.

The receive shadow register works similarly. Once a complete byte has been received, the receive shift register stores the newly received byte into the receive shadow register. The RDRF flag (or RXIFR flag if the received byte is part of an IFR) is set to indicate that a new byte of data has been received. The programmer has one BDLC module byte reception time to read the shadow register and clear the RDRF or RXIFR flag before the shadow register is overwritten by the newly received byte.

If the user writes the first byte of a message to be transmitted to the BDLC Data Register and then determines that a different message should be transmitted, the user can write a new byte to the BDLC Data Register up until the transmission begins. This new byte replaces the original byte in the BDLC Data Register.

From the time a byte is written to the BDLC Data Register until it is transferred to the transmit shift register, the transmit shadow register is considered full and the byte pending transmission. If one of the IFR transmission control bits (TSIFR, TMIFR1, or TMIFR0 in BDLC Control Register 2) is also set, the byte is pending transmission as an IFR. A byte pending transmission is flushed from the transmit shadow register and the transmission canceled if one of the following occurs: a loss of arbitration or transmitter error on the byte currently being transmitted; a symbol error, framing error, bus fault, or BREAK symbol is received. If the byte pending transmission is an IFR byte, the reception of a message with a CRC error also causes the byte in the transmit shadow register to be flushed.

To abort an in-progress transmission, the programmer should simply stop loading more data into the BDLC Data Register. This causes a transmitter underrun error and the BDLC module automatically disables the transmitter on the next non-byte boundary. This means that the earliest a transmission can be halted is after at least one byte (plus two extra 1-bits) has been transmitted. The receiver picks this up as an error and relay it in the state vector register as an invalid symbol error.

Offset + 0x05

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	D7	D6	D5	D4	D3	D2	D1	D0
W								
Reset	0	0	0	0	0	0	0	0

Figure 7-6. BDLC Data Register (DLCBDR)

Table 7-7. DLCBDR Field Descriptions

Field	Description
D[7:0]	Receive/Transmit Data

7.3.2.6 BDLC Analog Round Trip Delay Register (DLCBARD)

This register is used to program the BDLC module so that it compensates for the round trip delays of different external transceivers. Also the polarity of the receive pin (J1850_RX) is set in this register.

Read: any time

Write: write only once. Writes to unimplemented bits are ignored.

Offset + 0x08

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	RXPOL	0	BO4	BO3	BO2	BO1	BO0
W								
Reset	0	1	0	1	0	0	0	0

 = Unimplemented or Reserved

Figure 7-7. BDLC Analog Round Trip Delay Register (DLCBARD)

Table 7-8. DLCBARD Field Descriptions

Field	Description
RXPOL	<p>Receive Pin Polarity</p> <p>The Receive pin Polarity bit is used to select the polarity of incoming signal on the receive pin. Some external analog transceiver inverts the receive signal from the J1850 bus before feeding back to the digital receive pin.</p> <p>0 Select inverted polarity, where external transceiver inverts the receive signal.</p> <p>1 Select normal/true polarity; true non-inverted signal from J1850 bus, i.e., the external transceiver does not invert the receive signal.</p>

Offset + 0x08

Access: User read/write

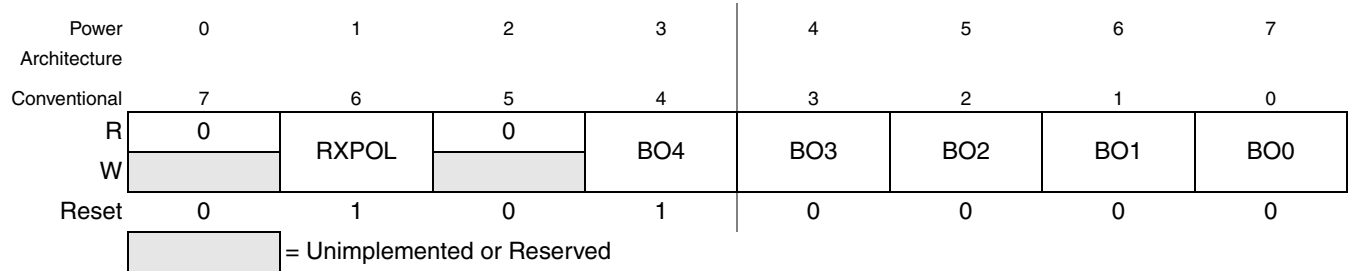


Figure 7-7. BDLC Analog Round Trip Delay Register (DLCBARD)

Table 7-8. DLCBARD Field Descriptions (continued)

Field	Description
BO[4:0]	<p>BDLC Analog Roundtrip Delay Offset Field</p> <p>Adjust the transmitted symbol timings to account for the differing roundtrip delays found in different SAE J1850 analog transceivers. The allowable delay range is from 0 μs to 31 μs, with a nominal target of 16 μs (reset value). Refer to Table for the BO[4:0] values corresponding to the expected transceiver delays and the resultant transmitter timing adjustment (in mux interface clock periods (t_{bdlc})). Refer to the analog transceiver device specification for the expected roundtrip delay through both the transmitter and the receiver. The sum of these two delays makes up the total roundtrip delay value.</p> <p>Note: For Digital Loopback test, the Analog Roundtrip Delay Offset Field should be set to 0μs.</p>

Table 7-9. BARD Values vs. Transceiver Delay and Transmitter Timing Adjustment

BARD Offset Bits BO[4:0]	Corresponding Expected Transceiver's delays (μ s)	Transmitter Symbol Timing Adjustment (t_{bdlc}^1)
00000	0	0
00001	1	1
00010	2	2
00011	3	3
00100	4	4
00101	5	5
00110	6	6
00111	7	7
01000	8	8
01001	9	9
01010	10	10
01011	11	11
01100	12	12
01101	13	13
01110	14	14
01111	15	15
10000	16	16
10001	17	17

Table 7-9. BARD Values vs. Transceiver Delay and Transmitter Timing Adjustment (continued)

BARD Offset Bits BO[4:0]	Corresponding Expected Transceiver's delays (μs)	Transmitter Symbol Timing Adjustment (t_{bdlc}^1)
10010	18	18
10011	19	19
10100	20	20
10101	21	21
10110	22	22
10111	23	23
11000	24	24
11001	25	25
11010	26	26
11011	27	27
11100	28	28
11101	29	29
11110	30	30
11111	31	31

¹ The transmitter symbol timing adjustment is the same for binary and integer bus frequencies.

7.3.2.7 BDLC Rate Select Register (DLCBRSR)

This register determines the divider prescaler value for the mux interface clock (f_{bdlc}). Only integer multiple of the 1 MHz or 1.048576 MHz f_{bdlc} are supported as input clock.

Read: any time

Write: write only once.

Offset + 0x09

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W								
Reset	0	0	0	0	0	0	0	0

Figure 7-8. BDLC Rate Select Register (DLCBRSR)

Table 7-10. DLCBRSR Field Descriptions

Field	Description
R[7:0]	Rate Select. These bits determine the amount by which the frequency of the system clock signal is divided to generate the MUX Interface clock (f_{bdlc}) which defines the basic timing resolution of the MUX Interface. The value programmed into these bits is dependent on the chosen system clock frequency. See Table 7-11 and Table 7-12 for example rate selects for different bus frequencies. All divisor values from divide by 1 to divide by 256 are possible, but are not shown in the tables. Note: Although the maximum divider is 256, a divider that generates a 1 MHz or 1.048576 MHz f_{bdlc} must be selected for J1850 communications to occur.

Table 7-11. BDLC Rate Selection for Binary Frequencies [CLKS = 1]

IP bus clock frequency	R[7:0]	division	f_{bdlc}
$f_{\text{CLOCK}}=1.048576$ MHz	0x00	1	1.048576 MHz

Table 7-12. BDLC Rate Selection for Integer Frequencies [CLKS = 0]

IP bus clock frequency	R[7:0]	division	f_{bdlc}
$f_{\text{CLOCK}}=66.00000$ MHz	0x41	66	1.000000 MHz
$f_{\text{CLOCK}}=54.00000$ MHz	0x35	54	1.000000 MHz
$f_{\text{CLOCK}}=33.00000$ MHz	0x20	33	1.000000 MHz
$f_{\text{CLOCK}}=27.00000$ MHz	0x1A	27	1.000000 MHz

7.3.2.8 BDLC Control Register (DLCSCR)

This register enables the BDLC module.

Read: any time

Write: any time

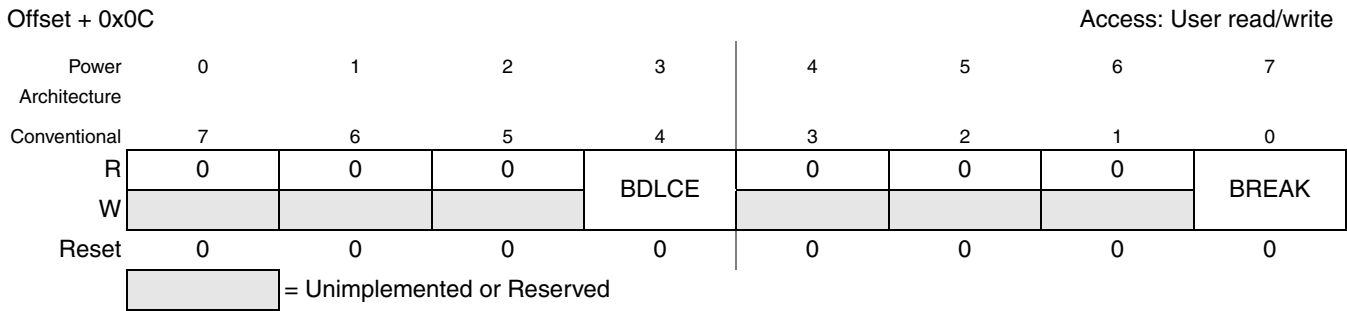


Figure 7-9. BDLC Control Register (DLCSCR)

Table 7-13. DLCSCR Field Descriptions

Field	Description
BDLCE	<p>BDLC Enable</p> <p>This bit serves as a mux interface clock (f_{bdlc}) enable/disable for power savings.</p> <p>1 The mux interface clock (f_{bdlc}) and BDLC module are enabled to allow J1850 communications to take place.</p> <p>0 The mux interface clock (f_{bdlc}) is disabled, shutting down the BDLC module for power saving. Bus clocks continue running, allowing registers to be accessed.</p>
BREAK	<p>Send BREAK signal</p> <p>This bit determines whether the BDLC module generates a BREAK symbol.</p> <p>0 The BDLC module does not generate a BREAK symbol.</p> <p>1 The BDLC module immediately sends a Break signal on the bus, regardless of its current transmit or receive status.</p> <p>After setting the BREAK bit it is automatically cleared after two IPB clock cycles.</p> <p>The active Break signal causes any other transmitting module to stop transmitting immediately because it loses arbitration. It is at least 280 μs long.</p> <p>Note: When the BDLC is operating at the high bus speed all 4X symbol times are one fourth that shown, except for Break, which is transmitted the same length in 1X or 4X mode.</p>

7.3.2.9 BDLC Status Register (DLCBSTAT)

This register indicates the status of the BLDC module.

Read: any time

Write: any time

Offset + 0x0D

Access: User read/write

Power PC	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TST_DIVTE_T4	TST_CRCV_T4	IDLE
W								
Reset	1	1	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 7-10. BDLC Status Register (DLCBSTAT)

Table 7-14. DLCBSTAT Field Descriptions

Field	Description
TST_DIVTE_T4	1 The module output Tx pin output mux interface clock (FBDLC). 0 The module output Tx pin output normal signal.
TST_CRCV_T4	Status of the receive message CRC 1 Incorrect 0 Correct
IDLE	This bit indicates when the BDLC module is idle. 0 BDLC module is either transmitting or receiving data. 1) BDLC module has received IFS and no data is being transmitted or received. NOTE BDLC module is only idle after receiving IFS. The IDLE bit is 0 during reset since the BDLC module needs to wait for an IFS before becoming idle. Noise on the bus will be filtered and the IDLE bit will remain unchanged.

7.4 Functional Description

The BDLC module is a serial communication module which allows the user to send and receive messages across a Society of Automotive Engineers (SAE) J1850 serial communication network. The user's software manages each transmitted or received message on a byte-by-byte basis, while the BDLC performs all of the network access, arbitration, message framing and error detection duties.

7.4.1 J1850 Frame Format

As noted above and in [Section 7.1.1, "Features"](#), the BDLC module communicates across an SAE J1850 network. As such, all messages transmitted on the J1850 bus are structured using the format below. The following sections describe this format and its meanings.

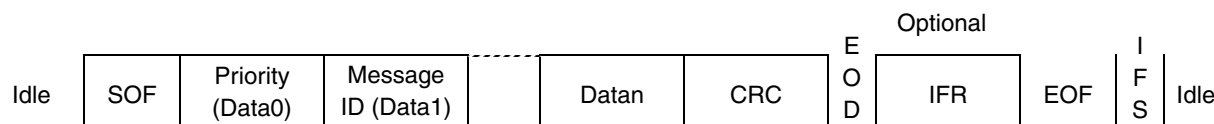


Figure 7-11. J1850 Bus Message Format (VPW)

SAE J1850 states that each message has a maximum length of 101 bit times or 12 bytes (excluding SOF, EOD, NB and EOF).

7.4.1.1 Start of Frame Symbol (SOF)

All messages transmitted onto the J1850 bus must begin with a long active SOF symbol. This indicates to any listeners on the J1850 bus the start of a new message transmission. The SOF symbol is not used in the CRC calculation.

7.4.1.2 In Message Data Bytes (Data)

The data bytes contained in the message include the message priority/type, message I.D. byte, and any actual data being transmitted to the receiving node. See SAE J1850 - Class B Data Communications Network Interface, for more information about 1 and 3 Byte Headers.

Messages transmitted by the BDLC module onto the J1850 bus must contain at least one data byte, and therefore can be as short as one data byte and one CRC byte. Each data byte in the message is 8 bits in length, transmitted MSB to LSB.

7.4.1.3 Cyclical Redundancy Check Byte (CRC)

This byte is used by the receiver(s) of each message to determine if any errors have occurred during the transmission of the message. The BDLC calculates the CRC byte and appends it onto any messages transmitted onto the J1850 bus, and also performs CRC detection on any messages it receives from the J1850 bus.

CRC generation uses the divisor polynomial $X^8+X^4+X^3+X^2+1$. The remainder polynomial is initially set to all ones, and then each byte in the message after the SOF symbol is serially processed through the CRC generation circuitry. The one's complement of the remainder then becomes the 8-bit CRC byte, which is appended to the message after the data bytes, in MSB to LSB order.

When receiving a message, the BDLC uses the same divisor polynomial. All data bytes, excluding the SOF and EOD symbols, but including the CRC byte, are used to check the CRC. If the message is error free, the remainder polynomial equals $X^7+X^6+X^2$ (0xC4), regardless of the data contained in the message. If the calculated CRC does not equal 0xC4, the BDLC recognizes this as a CRC error and sets the CRC error flag in the BDLC State Vector Register.

7.4.1.4 End-of-Data Symbol (EOD)

The EOD symbol is a long passive period on the J1850 bus used to signify to any recipients of a message that the transmission by the originator has completed. No flag is set upon reception of the EOD symbol.

7.4.1.5 In-Frame Response Bytes (IFR)

The IFR section of the J1850 message format is optional. Users desiring further definition of in-frame response should review the SAE J1850 Class B Data Communications Network Interface specification.

7.4.1.6 End-of-Frame Symbol (EOF)

This symbol is a passive period on the J1850 bus, longer than an EOD symbol, which signifies the end of a message. Because an EOF symbol is longer than an EOD symbol, if no response is transmitted after an

EOD symbol, it becomes an EOF, and the message is assumed to be completed. The EOF flag is set upon receiving the EOF symbol.

7.4.1.7 Inter-Frame Separation Symbol (IFS)

The IFS symbol is a passive period on the J1850 bus which allows proper synchronization between nodes during continuous message transmission. The IFS symbol is transmitted by a node following the completion of the EOF period.

When the last byte of a message has been transmitted onto the J1850 bus, and the EOF symbol time has expired, all nodes must then wait for the IFS symbol time to expire before transmitting an SOF, marking the beginning of another message.

However, if the BDLC module is waiting for the IFS period to expire before beginning a transmission and a rising edge is detected before the IFS time has expired, it internally synchronizes to that edge.

A rising edge may occur during the IFS period because of varying clock tolerances and loading of the J1850 bus, causing different nodes to observe the completion of the IFS period at different times. Receivers must synchronize to any SOF occurring during an IFS period to allow for individual clock tolerances.

7.4.1.8 Break

If the BDLC module is transmitting at the time a BREAK is detected, it treats the BREAK as if a transmission error had occurred, and halts transmission. The BDLC module can transmit a BREAK symbol. If while receiving a message the BDLC module detects a BREAK symbol, it treats the BREAK as a reception error and sets the invalid symbol flag. If while receiving a message in 4X mode, the BDLC module detects a BREAK symbol, it treats the BREAK as a reception error, sets BDLC State Vector Register register to 0x1C, and exits 4X mode. The 4XE bit in BDLC Control Register 2 is automatically cleared upon reception of the BREAK symbol.

7.4.1.9 Idle Bus

An idle condition exists on the bus during any passive period after expiration of the IFS period. Any node sensing an idle bus condition can begin transmission immediately.

7.4.2 J1850 VPW Symbols

Variable pulse width modulation (VPW) is an encoding technique in which each bit is defined by the time between successive transitions, and by the level of the bus between transitions, active or passive. Active and passive bits are used alternately. This encoding technique is used to reduced the number of bus transitions for a given bit rate. See [Section 7.1.1, “Features”](#).

The symbol values shown below are nominal values. Refer to the electrical specification for a more complete description of symbol values. Each logic one or logic zero contains a single transition, and can be at either the active or passive level and one of two lengths, either 64μs or 128μs (T_{NOM} at 10.4kbps baud rate), depending upon the encoding of the previous bit. The SOF, EOD, EOF and IFS symbols are always encoded at an assigned level and length. See [Figure 7-12](#).

Each message begins with an SOF symbol, an active symbol. Therefore, each data byte (including the CRC byte) begins with a passive bit, regardless of whether it is a logic one or a logic zero. All VPW bit lengths stated in the following descriptions are typical values at a 10.4 kbps bit rate.

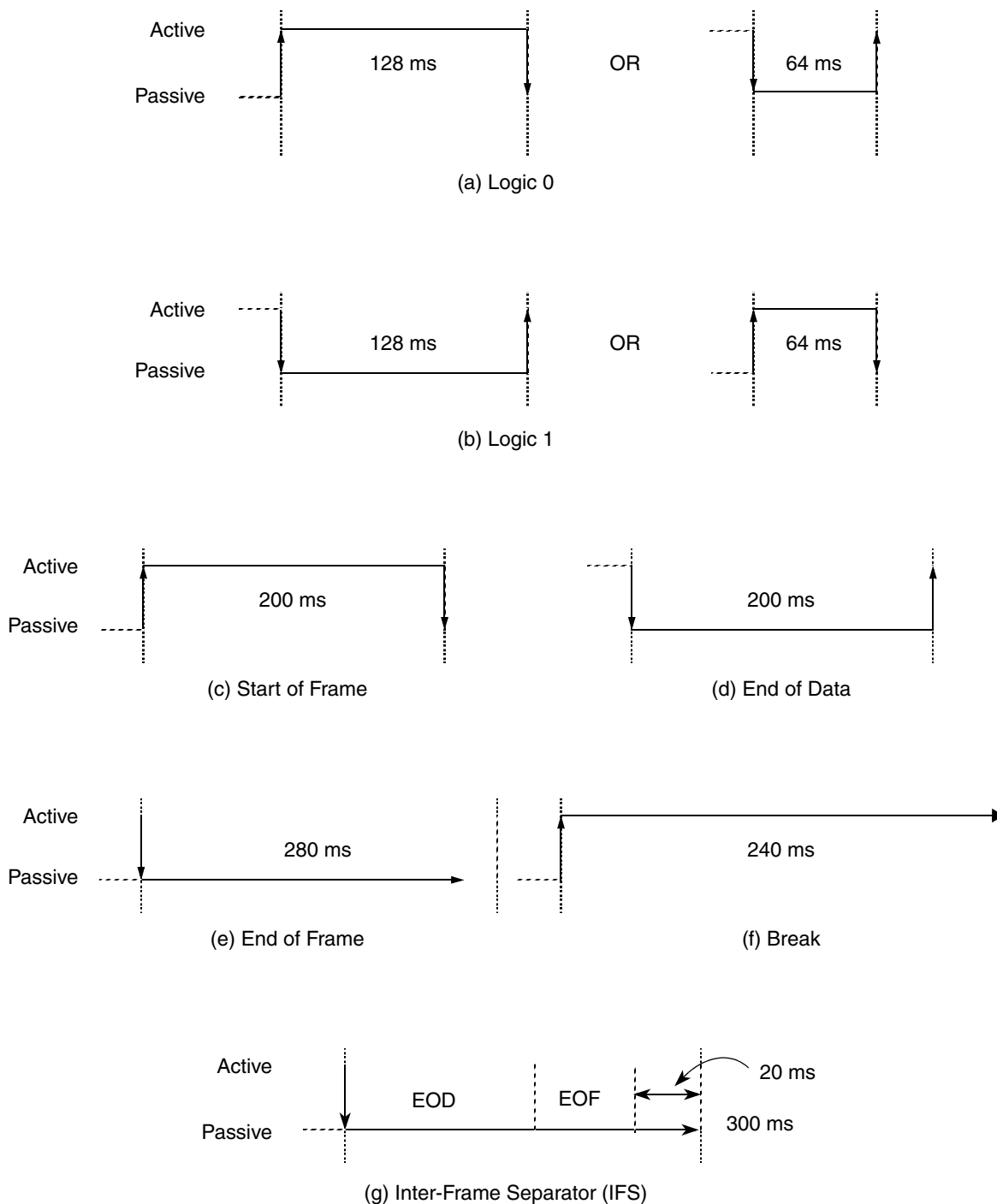


Figure 7-12. J1850 VPW Symbols

7.4.2.1 Logic 0

A logic zero is defined as either an active to passive transition followed by a passive period 64 μ s in length, or a passive to active transition followed by an active period 128 μ s in length (Figure 7-12(a)).

7.4.2.2 Logic 1

A logic one is defined as either an active to passive transition followed by a passive period 128 μ s in length, or a passive to active transition followed by an active period 64 μ s in length (Figure 7-12(b)).

7.4.2.3 Normalization Bit (NB)

The NB symbol has the same property as a logic 1 or a logic 0. It is only used in IFR message responses. This bit is defined as an active bit.

7.4.2.4 Start of Frame Symbol (SOF)

The SOF symbol is defined as passive to active transition followed by an active period 200 μ s in length (Figure 7-12(c)). This allows the data bytes which follow the SOF symbol to begin with a passive bit, regardless of whether it is a logic one or a logic zero.

7.4.2.5 End of Data Symbol (EOD)

The EOD symbol is defined as an active to passive transition followed by a passive period 200 μ s in length (Figure 7-12(d)).

7.4.2.6 End of Frame Symbol (EOF)

The EOF symbol is defined as an active to passive transition followed by a passive period 280 μ s in length (Figure 7-12(e)). If there is no IFR byte transmitted after an EOD symbol is transmitted, after another 80 μ s the EOD becomes an EOF, indicating the completion of the message.

7.4.2.7 Inter-Frame Separation Symbol (IFS)

The IFS symbol is defined as a passive period 300 μ s in length. The IFS symbol contains no transition, since when used it always follows an EOF symbol.(Figure 7-12(g))

7.4.2.8 Break Signal (BREAK)

The BREAK signal is defined as a passive to active transition followed by an active period of at least 240 μ s (Figure 7-12(f)).

7.4.2.9 IDLE

An IDLE is defined as a passive period greater than 300 μ s in length.

7.4.2.10 J1850 VPW Valid/Invalid Bits and Symbols

The timing tolerances for receiving data bits and symbols from the J1850 bus have been defined to allow for variations in oscillator frequencies. In many cases the maximum time allowed to define a data bit or symbol is equal to the minimum time allowed to define another data bit or symbol.

Because the minimum resolution of the BDLC module for determining which symbol is received equals a single period of the MUX Interface clock (t_{bdlc}), the receiver symbol timing boundaries are subject to an uncertainty of $1 t_{\text{bdlc}}$ due to sampling considerations.

This clock resolution of $1 t_{\text{bdlc}}$ allows the BDLC module to properly differentiate between the different bits and symbols, without reducing the valid window for receiving bits and symbols from transmitters onto the J1850 bus having varying oscillator frequencies.

7.4.2.10.1 Transmit and Receive Symbol Timing Specifications

Table 7-15 through Table 7-20 contain the SAE J1850 transmit and receive symbol timing specifications for the BDLC module. The units used in these tables are mux interface clock periods (t_{bdlc}). The mux interface clock is a divided down version of the bus clock input to the module (see Section 7.3.2.7, “BDLC Rate Select Register (DLCBRSR)”). The mux interface clock drives the transmit and receive counters which control symbol generation and identification. The symbol timing in effect during J1850 operations is dependent on the state of two control bits: the CLKS bit in the BDLC control register 1, which indicates whether the bus clock is an integer frequency or a binary frequency; the 4XE bit in BDLC Control Register 2, which is used to select 4X operation.

Table 7-15 and Table 7-17 indicate the transmit and receive timing for integer bus frequencies (CLKS = 0) and 4X operation disabled (4XE = 0). It is assumed that for integer bus frequencies the divided down mux interface clock frequency is 1 MHz ($t_{\text{bdlc}} = 1 \mu\text{s}$).

Table 7-16 and Table 7-18 indicated the transmit and receive timing for binary bus frequencies (CLKS = 1) and 4X operation disabled (4XE = 0). It is assumed that the divided down mux interface clock frequency is 1.048576 MHz ($t_{\text{bdlc}} = 0.953674 \mu\text{s}$) for binary bus frequencies. The symbol timing values are adjusted to compensate for the shortening of the mux interface clock period.

Table 7-19 and Table 7-20 show how the receive symbol timing values are adjusted when 4X operation is enabled (4XE = 1) for both integer bus frequencies (CLKS = 0) and binary bus frequencies (CLKS = 1), respectively.

The values specified in the tables are for the symbols appearing on the SAE J1850 bus. These values assume the BDLC module is communicating on the SAE J1850 bus using an external analog transceiver, and that the BDLC module analog roundtrip delay value programmed into the BDLC Analog Round Trip Delay Register register is the appropriate value for the transceiver being used. If these conditions are not met, the symbol timings being measured on the SAE J1850 bus are significantly affected. For a detailed description of how symbol timings are measured on the SAE J1850 bus, refer to the appropriate SAE documents.

Table 7-15. BDLC Transmitter VPW Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{tvp1}	62	64	66	t_{bdlc}
2	Passive Logic 1	T_{tvp2}	126	128	130	t_{bdlc}
3	Active Logic 0	T_{tva1}	126	128	130	t_{bdlc}
4	Active Logic 1	T_{tva2}	62	64	66	t_{bdlc}
5	Start of Frame (SOF)	T_{tva3}	198	200	202	t_{bdlc}
6	End of Data (EOD) ¹	T_{tvp3}	162	164	166	t_{bdlc}
7	End of Frame (EOF) ¹	T_{tv4}	238	240	242	t_{bdlc}
8	Inter-Frame Separator (IFS) ¹	T_{tv5}	298	300	302	t_{bdlc}

Note: The transmitter timing for this symbol depends upon the minimum detection time of the symbol by the receiver.

Table 7-16. BDLC Transmitter VPW Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{tvp1}	65	67	69	t_{bdlc}
2	Passive Logic 1	T_{tvp2}	132	134	136	t_{bdlc}
3	Active Logic 0	T_{tva1}	132	134	136	t_{bdlc}
4	Active Logic 1	T_{tva2}	65	67	69	t_{bdlc}
5	Start of Frame (SOF)	T_{tva3}	208	210	212	t_{bdlc}
6	End of Data (EOD) ¹	T_{tvp3}	170	172	174	t_{bdlc}
7	End of Frame (EOF) ¹	T_{tv4}	250	252	254	t_{bdlc}
8	Inter-Frame Separator (IFS) ¹	T_{tv5}	313	315	317	t_{bdlc}

¹ The transmitter timing for this symbol depends upon the minimum detection time of the symbol by the receiver.

Table 7-17. BDLC Receiver VPW Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	32	64	95	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	96	128	163	t_{bdlc}
3	Active Logic 0	T_{rva1}	96	128	163	t_{bdlc}
4	Active Logic 1	T_{rva2}	32	64	95	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	164	200	239	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	164	200	239	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	240	280	299	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	281	—	—	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	240	—	—	t_{bdlc}

Note: The receiver symbol timing boundaries are subject to an uncertainty of $1 t_{bdlc}$ due to sampling considerations.

Table 7-18. BDLC Receiver VPW Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	34	67	100	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	101	134	171	t_{bdlc}
3	Active Logic 0	T_{rva1}	101	134	171	t_{bdlc}
4	Active Logic 1	T_{rva2}	34	67	100	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	172	210	251	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	172	210	251	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	252	293	314	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	315	—	—	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	252	—	—	t_{bdlc}

Note: The receiver symbol timing boundaries are subject to an uncertainty of $1 t_{bdlc}$ due to sampling considerations.

Table 7-19. BDLC Receiver VPW 4X Symbol Timing for Integer Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	8	16	23	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	24	32	40	t_{bdlc}
3	Active Logic 0	T_{rva1}	24	32	40	t_{bdlc}
4	Active Logic 1	T_{rva2}	8	16	23	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	41	50	59	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	41	50	59	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	60	70	74	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	75	—	—	t_{bdlc}
9	Break Signal (BREAK)	T_{rv6}	60	—	—	t_{bdlc}

Note: The receiver symbol timing boundaries are subject to an uncertainty of $1 t_{bdlc}$ due to sampling considerations.

Table 7-20. BDLC Receiver VPW 4X Symbol Timing for Binary Frequencies

Number	Characteristic	Symbol	Min	Typ	Max	Unit
1	Passive Logic 0	T_{rvp1}	9	17	25	t_{bdlc}
2	Passive Logic 1	T_{rvp2}	26	34	42	t_{bdlc}
3	Active Logic 0	T_{rva1}	26	34	42	t_{bdlc}
4	Active Logic 1	T_{rva2}	9	17	25	t_{bdlc}
5	Start of Frame (SOF)	T_{rva3}	43	53	62	t_{bdlc}
6	End of Data (EOD)	T_{rvp3}	43	53	62	t_{bdlc}
7	End of Frame (EOF)	T_{rv4}	63	74	78	t_{bdlc}
8	Inter-Frame Separator (IFS)	T_{rv5}	79	—	—	t_{bdlc}

Table 7-20. BDLC Receiver VPW 4X Symbol Timing for Binary Frequencies

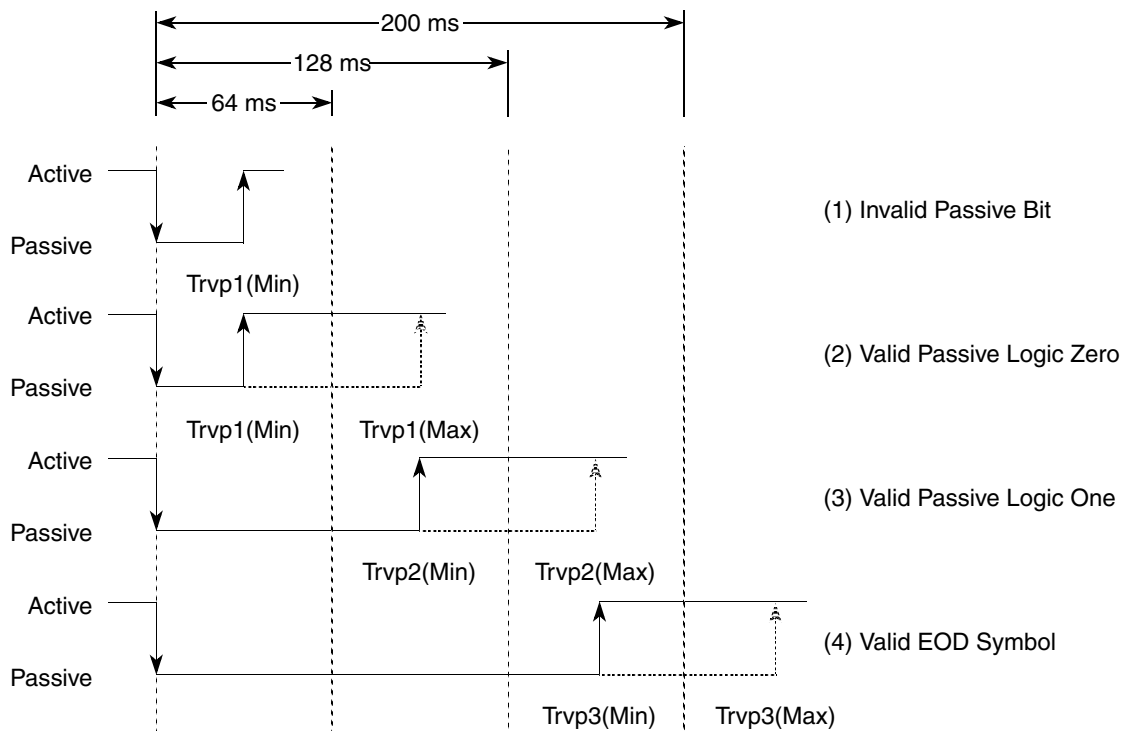
Number	Characteristic	Symbol	Min	Typ	Max	Unit
9	Break Signal (BREAK)	T_{rv6}	63	—	—	t_{bdlc}

Note: The receiver symbol timing boundaries are subject to an uncertainty of $1 t_{bdlc}$ due to sampling considerations.

The minimum and maximum symbol limits shown in the following sections ([Invalid Passive Bit–Valid BREAK Symbol](#)) and figures ([Figure 7-13](#) - [Figure 7-16](#)) refer to the values listed in [Table 7-15](#) through [Table 7-20](#).

Invalid Passive Bit

If the passive to active transition beginning the next data bit or symbol occurs between the active to passive transition beginning the current data bit or symbol and $T_{rvp1}(\text{Min})$, the current bit would be invalid. See [Figure 7-13\(1\)](#).

**Figure 7-13. J1850 VPW Passive Symbols**

Valid Passive Logic Zero

If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp1}(\text{Min})$ and $T_{rvp1}(\text{Max})$, the current bit would be considered a logic zero. See [Figure 7-13\(2\)](#).

Valid Passive Logic One

If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp2}(\text{Min})$ and $T_{rvp2}(\text{Max})$, the current bit would be considered a logic one. See [Figure 7-13\(3\)](#).

Valid EOD Symbol

If the passive to active transition beginning the next data bit or symbol occurs between $T_{rvp3}(\text{Min})$ and $T_{rvp3}(\text{Max})$, the current symbol would be considered a valid EOD symbol. See Figure 7-13(4).

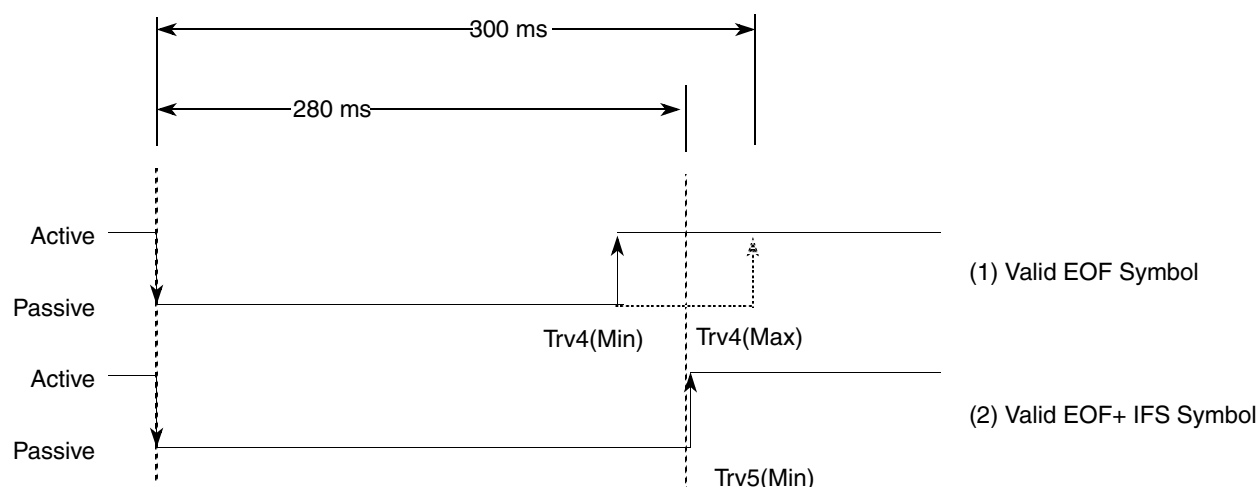


Figure 7-14. J1850 VPW EOF and IFS Symbols

Valid EOF and IFS Symbol

In Figure 7-14(1), if the passive to active transition beginning the SOF symbol of the next message occurs between $T_{rv4}(\text{Min})$ and $T_{rv4}(\text{Max})$, the current symbol is considered a valid EOF symbol.

If the passive to active transition beginning the SOF symbol of the next message occurs after $T_{rv5}(\text{Min})$, the current symbol is considered a valid EOF symbol followed by a valid IFS symbol. See Figure 7-14(2). All nodes must wait until a valid IFS symbol time has expired before beginning transmission. However, due to variations in clock frequencies and bus loading, some nodes may recognize a valid IFS symbol before others, and immediately begin transmitting. Therefore, anytime a node waiting to transmit detects a passive to active transition once a valid EOF has been detected, it should immediately begin transmission, initiating the arbitration process.

Idle Bus

If the passive to active transition beginning the SOF symbol of the next message does not occur before $T_{rv5}(\text{Min})$, the bus is considered to be idle, and any node wishing to transmit a message may do so immediately.

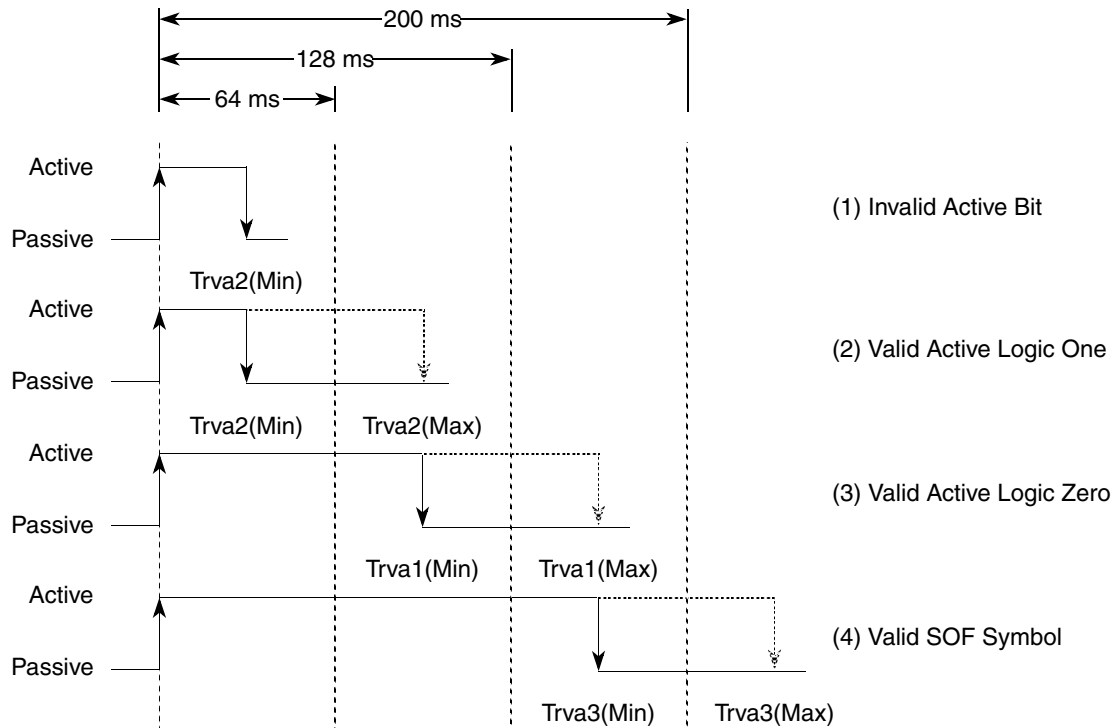


Figure 7-15. J1850 VPW Active Symbols

Invalid Active Bit

If the active to passive transition beginning the next data bit or symbol occurs between the passive to active transition beginning the current data bit or symbol and $T_{rva2(Min)}$, the current bit would be invalid. See Figure 7-15(1).

Valid Active Logic One

If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva2(Min)}$ and $T_{rva2(Max)}$, the current bit would be considered a logic one. See Figure 7-15(2).

Valid Active Logic Zero

If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva1(Min)}$ and $T_{rva1(Max)}$, the current bit would be considered a logic zero. See Figure 7-15(3).

Valid SOF Symbol

If the active to passive transition beginning the next data bit or symbol occurs between $T_{rva3(Min)}$ and $T_{rva3(Max)}$, the current symbol would be considered a valid SOF symbol. See Figure 7-15(4).

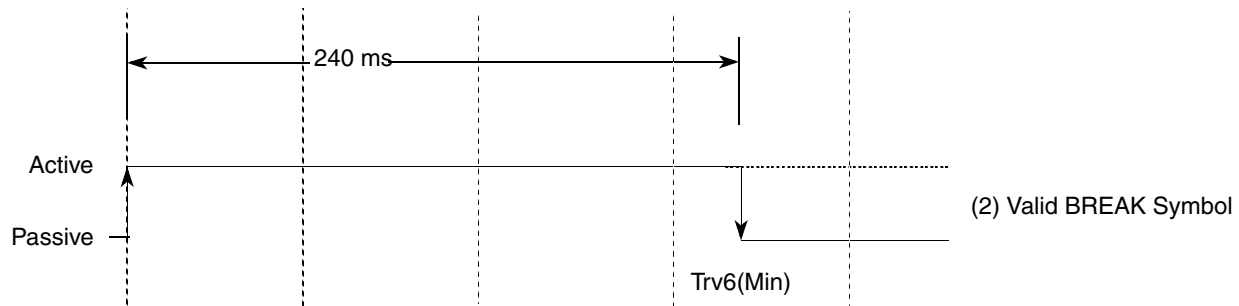


Figure 7-16. J1850 VPW BREAK Symbol

Valid BREAK Symbol

If the next active to passive transition does not occur until after $T_{rv6}(\text{Min})$, the current symbol is considered a valid BREAK symbol. A BREAK symbol should be followed by a SOF symbol beginning the next message to be transmitted onto the J1850 bus. See [Figure 7-16](#).

7.4.2.10.2 Message Arbitration

Message arbitration on the J1850 bus is accomplished in a non-destructive manner, allowing the message with the highest priority to be transmitted, while any transmitters which lose arbitration simply stop transmitting and wait for an idle bus to begin transmitting again.

If the BDLC module wishes to transmit onto the J1850 bus, but detects that another message is in progress, it automatically waits until the bus is idle. However, if multiple nodes begin to transmit in the same synchronization window, message arbitration occurs beginning with the first bit after the SOF symbol and continue with each bit thereafter.

The VPW symbols and J1850 bus electrical characteristics are carefully chosen so that a logic zero (active or passive type) always dominates over a logic one (active or passive type) simultaneously transmitted. Hence logic zeroes are said to be dominant and logic ones are said to be recessive.

When a node transmits a recessive bit and detects a dominant bit, it loses arbitration, and immediately stops transmitting. This is known as bitwise arbitration. The loss of arbitration flag (in BDLC State Vector Register) is set when arbitration is lost. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

During arbitration, or even throughout the transmitting message, when an opposite bit is detected, transmission is immediately stopped unless it occurs on the 8th bit of a byte. In this case, the BDLC module automatically appends up to two extra 1 bits and then stop transmitting. These two extra bits are arbitrated normally and thus do not interfere with another message. The second 1 bit is not sent if the first loses arbitration. If the BDLC module has lost arbitration to another valid message, the two extra ones do not corrupt the current message. However, if the BDLC module has lost arbitration due to noise on the bus, the two extra ones ensure the current message is detected and ignored as a noise-corrupted message.

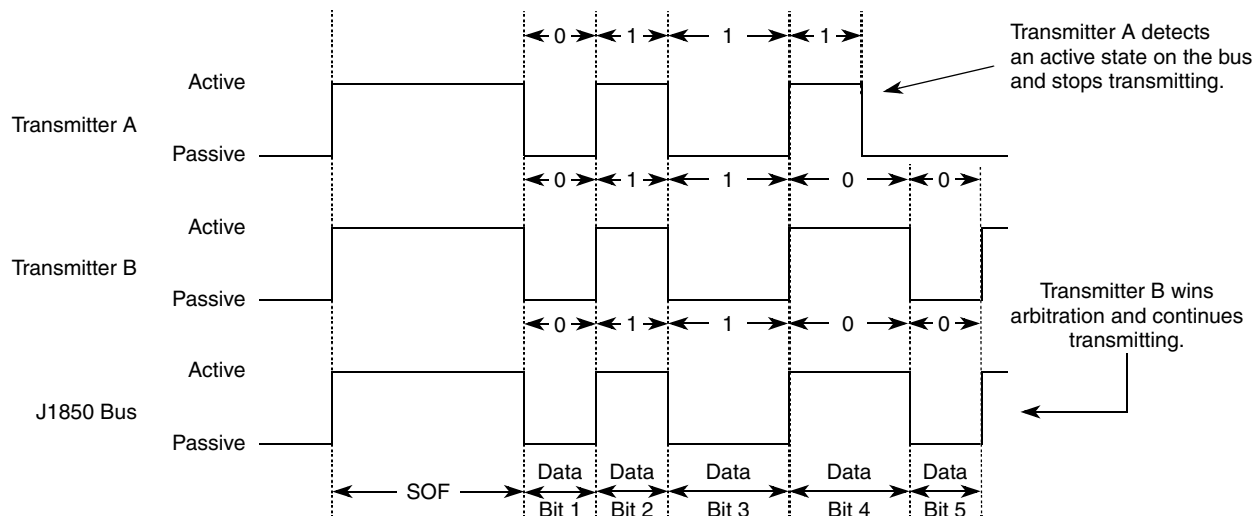


Figure 7-17. J1850 VPW Bitwise Arbitrations

Because a 0 dominates a 1, the message with the lowest value has the highest priority and always wins arbitration. A message with priority 000 wins arbitration over a message with priority 011. This method of arbitration works no matter how many bits of priority encoding are contained in the message.

7.4.2.11 J1850 Bus Errors

The BDLC module detects several types of transmit and receive errors which can occur during the transmission of a message onto the J1850 bus.

7.4.2.11.1 Transmission Error

If the BDLC module is transmitting a message and the message received contains a symbol error, a framing error, a bus fault, a BREAK symbol, or a logic 1 symbol when a logic 0 is being transmitted, this constitutes a transmission error. Receiving a logic 0 symbol when transmitting a logic 1 is considered a loss of arbitration condition (See [Section 7.4.2.10.2, “Message Arbitration”](#)) and not a transmission error. When a transmission error is detected, the BDLC module immediately ceases transmitting. Further transmission or reception is disabled until a valid EOF symbol is detected on the J1850 bus. The error condition is reflected by setting the symbol invalid or out of range flag in the BDLC State Vector Register register. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

7.4.2.11.2 CRC Error

A cyclical redundancy check (CRC) error is detected when the data bytes and CRC byte of a received message are processed, and the CRC calculation result is not equal to 0xC4. The CRC code should detect any single and 2 bit errors, as well as all 8 bit burst errors, and almost all other types of errors. The CRC error flag (in BDLC State Vector Register) is set when a CRC error is detected. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

7.4.2.11.3 Symbol Error

A symbol error is detected when an abnormal (invalid) symbol is detected in a message being received from the J1850 bus. See [Invalid Passive Bit](#) and [Invalid Active Bit](#) which define invalid symbols. The symbol invalid or out of range flag (in BDLC State Vector Register) is set when a symbol error is detected. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

7.4.2.11.4 Framing Error

A framing error is detected when a received symbol occurs in an inappropriate location in the message frame. The following situations result in framing errors:

- An active logic 0 or logic 1 received as the first symbol of the frame.
- An SOF symbol received in any location other than the first symbol of a frame. Erroneous locations include: Within the data portion of a message or IFR; Immediately following the EOD in a message or IFR.
- An EOD symbol received on a non-byte boundary in a message or IFR.
- An active logic 0 or logic 1 received immediately following the EOD at the end of an IFR.

The symbol invalid or out of range flag (in BDLC State Vector Register) is set when a framing error is detected. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

7.4.2.11.5 Bus Fault

If a bus fault occurs, the response of the BDLC module depends upon the type of bus fault.

If the bus is shorted to V_{DD} , the BDLC module waits for the bus to fall to a passive state before it attempts to transmit a message. As long as the short remains, the BDLC never attempts to transmit a message onto the J1850 bus.

If the bus is shorted to ground, the BDLC module sees an idle bus, begin to transmit the message, and then detect a transmission error, since the short to ground would not allow the bus to be driven to the active (dominant) state. The BDLC module waits for assertion of the receive pin for (64 - analog round trip delay) t_{bdlc} cycles, after assertion of the transmit pin, before detecting the error. If the transmission is an IFR, the BDLC module waits for (280 - analog round trip delay) t_{bdlc} cycles before detecting an error. The analog round trip delay is determined by the value stored in the BDLC Analog Round Trip Delay Register register. The BDLC module sets the symbol invalid or out of range flag (in BDLC State Vector Register), abort that transmission and wait for the next CPU command to transmit. In this case, the transmitter does not have to wait for an EOF symbol to be received to be enabled. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

If the bus fault is temporary, as soon as the fault is cleared, the BDLC module resumes normal operation. If the bus fault is permanent, it may result in permanent loss of communication on the J1850 bus.

7.4.2.11.6 Break

Any BDLC transmitting at the time a BREAK is detected treats the BREAK as if a transmission error had occurred, and halt transmission.

If while receiving a message the BDLC module detects a BREAK symbol, it treats the BREAK as a reception error.

If a BREAK symbol is received while the BDLC module is transmitting or receiving, the symbol invalid or out of range flag (in BDLC State Vector Register) is set. Further transmission/reception is disabled until the J1850 bus returns to the passive state and a valid EOF symbol is detected on the J1850 bus. If the interrupt enable bit (IE in BDLC Control Register 1) is set, an interrupt request from the BDLC module is generated. Reading the BDLC State Vector Register register clears this flag.

The BDLC module can transmit a BREAK symbol. It can receive a BREAK symbol from the J1850 bus.

7.4.2.12 Bus Error Summary

The possible J1850 bus errors and the actions taken by the BDLC module are summarized in [Table 7-21](#).

Table 7-21. BDLC Module J1850 Error Summary

Error Condition	BDLC Module Function
Transmission Error	BDLC module immediately ceases transmitting. Further transmission and reception is disabled until a valid EOF symbol is detected. The symbol invalid or out of range flag is set and interrupt generated if enabled.
Cyclical Redundancy Check (CRC) Error	CRC error flag set and interrupt generated if enabled.
Symbol Error	The symbol invalid or out of range flag is set and interrupt generated if enabled. Transmission and reception is disabled until a valid EOF symbol is detected.
Framing Error	The symbol invalid or out of range flag is set and interrupt generated if enabled. Transmission and reception is disabled until a valid EOF symbol is detected.
Bus short to V _{DD} .	The BDLC module does not transmit until short is corrected and a valid EOF is detected. Depending upon when short occurs and is corrected, this error condition may set the symbol invalid or out of range, crc error, or loss of arbitration flags.
Bus short to GND.	Short is seen as an idle bus by BDLC module. If a transmission attempt is made before short is corrected, the symbol invalid or out of range flag is set and interrupt generated if enabled. Another transmission can be initiated as soon as short is corrected.
BREAK symbol reception	If doing so, the BDLC module immediately ceases transmitting. Symbol invalid or out of range flag set and interrupt generated if enabled. Transmission and reception is disabled until a valid EOF symbol is detected.

7.4.3 MUX Interface

The MUX Interface is responsible for bit encoding/decoding and digital noise filtering between the Protocol Handler and the Physical Interface. Refer to [Figure 7-1](#).

7.4.3.1 Mux Interface – Rx Digital Filter

The Receiver section of the BDLC module includes a digital low pass filter to remove narrow noise pulses from the incoming message. An outline of the digital filter is shown in Figure 7-18.

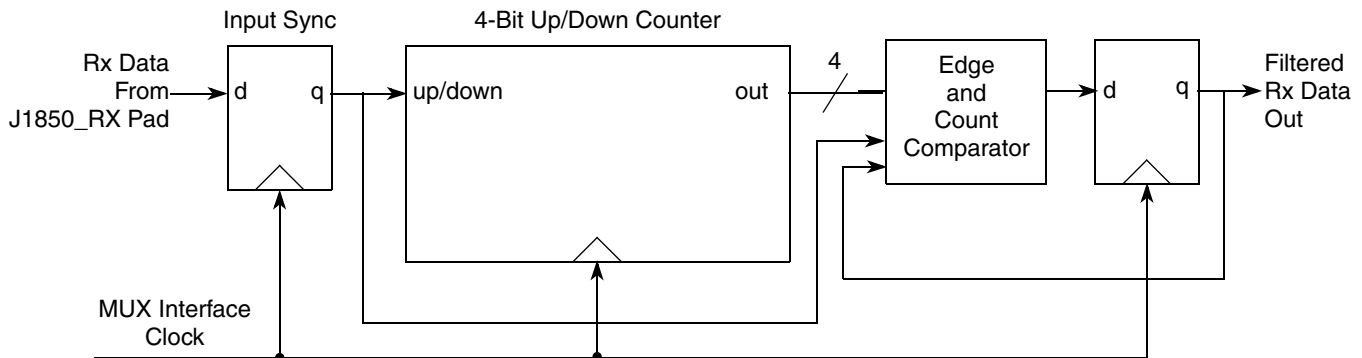


Figure 7-18. BDLC Module Rx Digital Filter Block Diagram

7.4.3.2 Operation

The clock for the digital filter is provided by the MUX Interface clock. At each positive edge of the clock signal, the current state of the Receiver input signal from the J1850_RX pad is sampled. The J1850_RX signal state is used to determine whether the counter should increment or decrement at the next positive edge of the clock signal.

The counter increments if the input data sample is high but decrement if the input sample is low. The counter then progresses up towards 15 if, on average, the J1850_RX signal remains high or progress down towards 0 if, on average, the J1850_RX signal remains low.

When the counter eventually reaches the value 15, the digital filter decides that the condition of the J1850_RX signal is at a stable logic level one and the Data Latch is set, causing the Filtered Rx Data signal to become a logic level one. Furthermore, the counter is prevented from overflowing and can only be decremented from this state.

Alternatively, should the counter eventually reach the value 0, the digital filter decides that the condition of the J1850_RX signal is at a stable logic level zero and the Data Latch is reset, causing the Filtered Rx Data signal to become a logic level zero. Furthermore, the counter is prevented from underflowing and can only be incremented from this state.

The Data Latch retains its value until the counter next reaches the opposite end point, signifying a definite transition of the J1850_RX signal.

7.4.3.3 Performance

The performance of the digital filter is best described in the time domain rather than the frequency domain.

If the signal on the J1850_RX signal transitions, there is a delay before that transition appears at the Filtered Rx Data output signal. This delay is between 15 and 16 clock periods, depending on where the transition occurs with respect to the sampling points. This filter delay must be taken into account when performing message arbitration.

For example, if the frequency of the MUX Interface clock (f_{bdlc}) is 1.0486MHz, then the period (t_{bdlc}) is 954ns and the maximum filter delay in the absence of noise is 15.259us.

The effect of random noise on the J1850_RX signal depends on the characteristics of the noise itself. Narrow noise pulses on the J1850_RX signal is completely ignored if they are shorter than the filter delay. This provides a degree of low pass filtering.

If noise occurs during a symbol transition, the detection of that transition may be delayed by an amount equal to the length of the noise burst. This is a reflection of the uncertainty of where the transition is truly occurring within the noise.

Noise pulses that are wider than the filter delay, but narrower than the shortest allowable symbol length is detected by the next stage of the BDLC module's receiver as an invalid symbol.

Noise pulses that are longer than the shortest allowable symbol length is normally detected as an invalid symbol or as invalid data when the frame's CRC is checked.

7.4.4 Protocol Handler

The Protocol Handler is responsible for framing, collision detection, arbitration, CRC generation/checking, and error detection. The Protocol Handler conforms to SAE J1850 - Class B Data Communications Network Interface. Refer to [Figure 7-1](#).

7.4.4.1 Protocol Architecture

The Protocol Handler contains the State Machine, Rx Shadow Register, Tx Shadow Register, Rx Shift Register, Tx Shift Register, and Loopback Multiplexer as shown in [Figure 7-19](#).

7.4.4.1.1 Rx and Tx Shift Registers

The Rx Shift Register gathers received serial data bits from the J1850 bus and makes them available in parallel form to the Rx Shadow Register. The Tx Shift Register takes data, in parallel form, from the Tx Shadow Register and presents it serially to the State Machine so that it can be transmitted onto the J1850 bus.

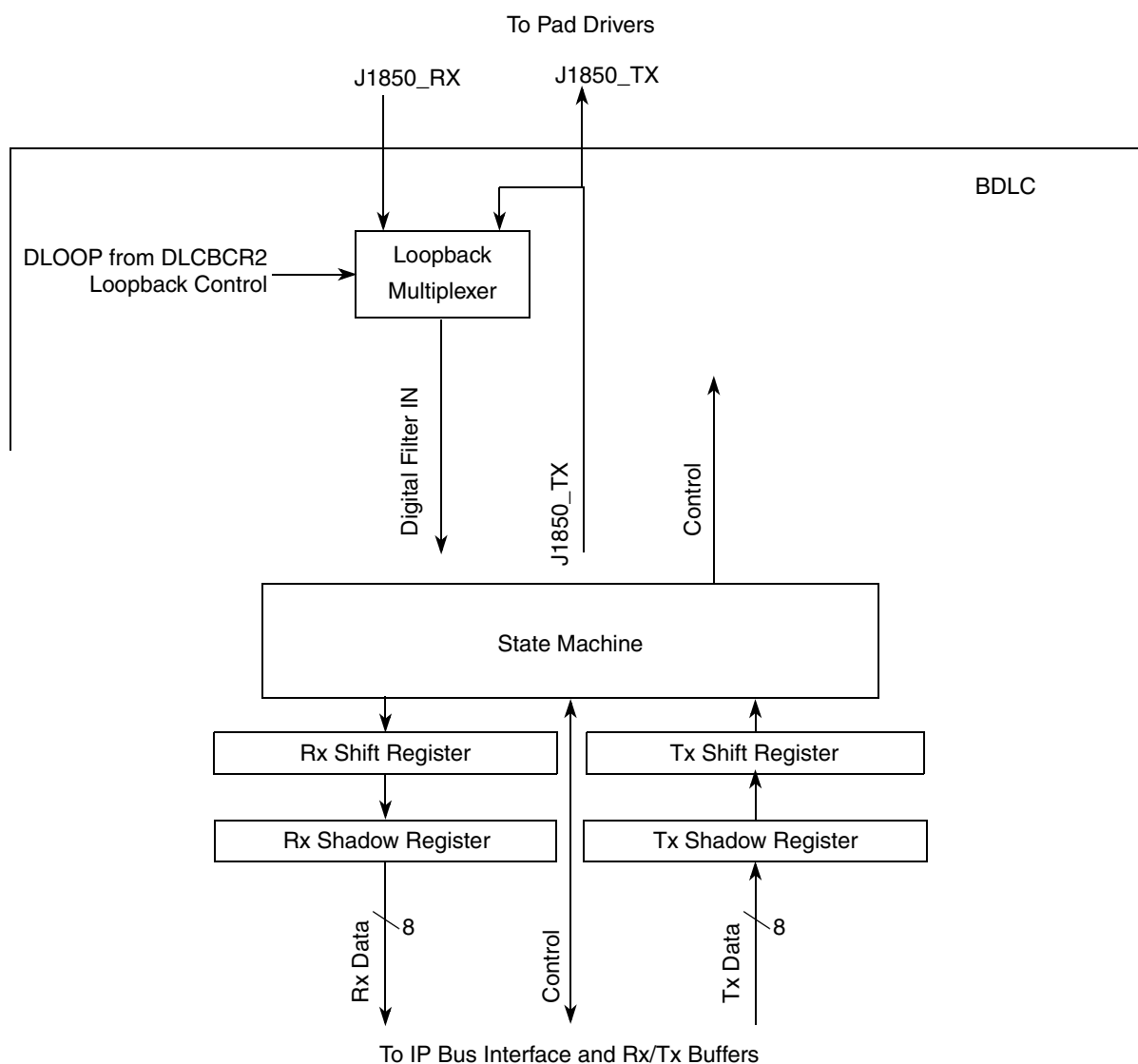


Figure 7-19. BDLC Protocol Handler Outline

7.4.4.1.2 Rx and Tx Shadow Registers

Immediately after the Rx Shift Register has completed shifting in a byte of data, this data is transferred to the Rx Shadow Register and RDRF or RXIFR is set and interrupt is generated if the interrupt enable bit (IE) in BDLC Control Register 1 is set. After the transfer takes place, this new data byte in the Rx Shadow Register is available to the CPU, and the Rx Shift Register is ready to shift in the next byte of data. Data in Rx Shadow Register must be retrieved by the CPU before it is overwritten by new data from the Rx Shift Register.

After the Tx Shift Register has completed its shifting operation for the current byte, the data byte in the Tx Shadow Register is loaded into the Tx Shift Register. After this transfer takes place, the Tx Shadow Register is ready to accept new data from the CPU.

7.4.4.1.3 Digital Loopback Multiplexer

The digital loopback multiplexer connects the input of the receive digital filter (See [Figure 7-19](#)) to either the transmit signal out to the pad (J1850_TX) or the receive signal from the pad (J1850_RX), depending on the DLOOP bit in BDLC Control Register 2 register.

7.4.4.1.4 State Machine

All of the functions associated with performing the protocol are executed or controlled by the State Machine. The State Machine is responsible for framing, collision detection, arbitration, CRC generation/checking, and error detection. The following sections describe the BDLC module's actions in a variety of situations.

7.4.4.1.5 4X Mode

The BDLC module can exist on the same J1850 bus as modules that use a special 4X (41.6 kbps) mode of J1850 VPW operation. The BDLC module can transmit and receive messages in 4X mode, if the 4XE bit is set in BDLC Control Register 2. If the 4XE bit is not set in the BDLC Control Register 2, any 4X message on the J1850 bus is treated as noise by the BDLC module and is ignored. Likewise, 4X messages transmitted on the SAE J1850 bus when the BDLC module is in normal mode is interpreted as noise on the network by the BDLC module.

7.4.4.1.6 Receiving a Message in Block Mode

Although not a part of the SAE J1850 protocol, the BDLC module allows for a special block mode of operation for the receiver. As far as the BDLC module is concerned, a block mode message is simply a long J1850 frame that contains an indefinite number of data bytes. All of the other features of the frame remain the same, including the SOF, CRC, and EOD symbols.

Another node wishing to send a block mode transmission must first inform all other nodes on the network that this is about to happen. This is usually accomplished by sending a special predefined message.

7.4.4.1.7 Transmitting a Message in Block Mode

A Block mode message is transmitted inherently by simply loading the bytes one by one into the BDLC Data Register register until the message is complete. The programmer should wait until the TDRE flag is set prior to writing a new byte of data into the BDLC Data Register register. The BDLC module does not contain any predefined maximum J1850 message length requirement.

7.4.5 Transmitting A Message

The design of the BDLC module enables you to manage message reception and message transmission separately. All received messages can be managed almost identically, regardless of their origin.

This chapter only describes the steps necessary for transmitting a message and does not address the resulting reception of that message by the BDLC module. Message reception is described in [Section 7.4.6, "Receiving A Message"](#). Later sections deal with transmitting and receiving In-Frame Responses on the SAE J1850 bus.

7.4.5.1 BDLC Transmission Control Bits

There is only one BDLC module control bit which is used when transmitting a message onto the SAE J1850 bus. This bit, the Transmit End of Data (TEOD) bit, is set by the user to indicate to the BDLC module that the last byte of that part of the message frame has been loaded into the BDLC Data Register. The TEOD bit, located in BDLC Control Register 2, is also used when transmitting an In-Frame Response (IFR), but that usage is described in [Section 7.4.7, “Transmitting an In-Frame Response \(IFR\)”](#). Setting the TEOD bit indicates to the BDLC module that the last byte written to the BDLC Data Register is the final byte to be transmitted, and that following this byte a CRC byte and EOD symbol should be transmitted automatically. Setting the TEOD bit also inhibits any further TDRE interrupts until TEOD is cleared. The TEOD bit is cleared on the rising edge of the first bit of the transmitted CRC byte, or if an error or loss of arbitration is detected on the bus.

7.4.5.1.1 BDLC Data Register

The BDLC data register is a double-buffered register which is used for handling the transmitted and received message bytes. Bytes to be transmitted onto the SAE J1850 bus are written to the BDLC data register, and bytes received from the bus by the BDLC module are read from the BDLC data register. Because this register is double buffered, bytes written into it cannot be read by the CPU. If this is attempted, the read byte is the last byte placed in the BDLC data register by the BDLC module, not the last byte written to the BDLC data register by the CPU. For an illustration of the BDLC data register, refer to [Section 7.3.2.5, “BDLC Data Register \(DLCBDR\)”](#).

7.4.5.1.2 Transmitting a Message with the BDLC

To transmit a message using the BDLC module, the user writes the first byte of the message to be transmitted into the BDLC Data Register, initiating the transmission process. When the TDRE status appears in the BDLC State Vector Register, the user writes the next byte into the BDLC Data Register. After all of the bytes have been loaded into the BDLC Data Register, the user sets the TEOD bit, and the BDLC module completes the message transmission. What follows is an overview of the basic steps required to transmit a message onto an SAE J1850 network using the BDLC module. For an illustration of this sequence, refer to [Figure 7-20](#).

NOTE

Due to the byte-level architecture of the BDLC module, the 12-byte limit on message length as defined in SAE J1850 must be enforced by the user's software. The number of bytes in a message (transmitted or received) has no meaning to the BDLC module.

1. Write the First Byte into the BDLC Data Register

To initiate a message transmission, the CPU simply loads the first byte of the message to be transmitted into the BDLC Data Register. The BDLC module then performs the necessary bus acquisition duties to determine when the message transmission can begin.

After the BDLC module determines that the SAE J1850 bus is free, a Start of Frame (SOF) symbol is transmitted, followed by the byte written to the BDLC Data Register. After the BDLC module readies this byte for transmission, the BDLC State Vector Register reflects that the next byte can be written to the BDLC Data Register (TDRE interrupt).

NOTE

If the user writes the first byte of a message to be transmitted to the BDLC Data Register and then determines that a different message should be transmitted, the user can write a new byte to the BDLC Data Register up until the transmission begins. This new byte replaces the original byte in the BDLC Data Register.

2. When TDRE is Indicated, Write the Next Byte into the BDLC Data Register

When a TDRE state is reflected in the BDLC State Vector Register, the CPU writes the next byte to be transmitted into the BDLC Data Register. This step is repeated until the last byte to be transmitted is written to the BDLC Data Register.

NOTE

Due to the design and operation of the BDLC module, when transmitting a message the user may write two, or possibly even three of the bytes to be transmitted into the BDLC Data Register before the first RDRF interrupt occurs. For this reason, the user should never use receive interrupts to control the sequencing of bytes to be transmitted.

3. Write the Last Byte to the BDLC Data Register and Set TEOD

After the user has written the last byte to be transmitted into the BDLC Data Register, the user then sets the TEOD bit in BDLC Control Register 2. When the TEOD bit is set, once the byte written to the BDLC Data Register is transmitted onto the bus, the BDLC module begins transmitting the 8-bit CRC byte, as specified in SAE J1850. Following the CRC byte, the BDLC module transmits an EOD symbol onto the SAE J1850 bus, indicating that this part of the message has been completed. If no IFR bytes are transmitted following the EOD, an EOF is recognized and the message is complete.

Setting the TEOD bit is the last step the CPU needs to take to complete the message transmission, and no further transmission-related interrupts occur. After the message has been completely received by the BDLC module, an EOF interrupt is generated. However, this is technically a receive function that can be managed by the message reception routine.

NOTE

While the TEOD bit is typically set immediately following the write of the last byte to the BDLC Data Register, it is also acceptable to wait until a TDRE interrupt is generated before setting the TEOD bit. While the example flowchart in [Figure 7-20](#) shows the TEOD bit being set after the write to the BDLC Data Register, either method is correct. If a TDRE interrupt is pending, it is cleared when the TEOD bit is set.

7.4.5.2 Transmitting Exceptions

While this is the basic transmit flow, at times the message transmit process is interrupted. This can be due to a loss of arbitration to a higher priority message or due to an error being detected on the network. For the transmit routine, either of these events can be dealt with in a similar manner.

7.4.5.2.1 Loss of Arbitration

If a loss of arbitration (LOA) occurs while the BDLC module is transmitting onto the SAE J1850 bus, the BDLC module immediately stops transmitting, and a LOA status is reflected in the BDLC State Vector Register. If the loss of arbitration has occurred on a byte boundary, an RDRF interrupt may also be pending once the LOA interrupt is cleared.

When a loss of arbitration occurs, the J1850 message handling software should immediately switch into the receive mode. If the TEOD bit was set, it is cleared automatically. If another attempt is to be made to transmit the same message, the user must start the transmit sequence over from the beginning of the message.

7.4.5.2.2 Error Detection

Similar to a loss of arbitration, if any error (except a CRC error) is detected on the SAE J1850 bus during a transmission, the BDLC module stops transmitting immediately. The transmitted byte is discarded, and the symbol invalid or out of range status is reflected in the BDLC State Vector Register. As with the loss of arbitration, if the TEOD bit was set, it is cleared automatically and any attempt to transmit the same message has to start from the beginning.

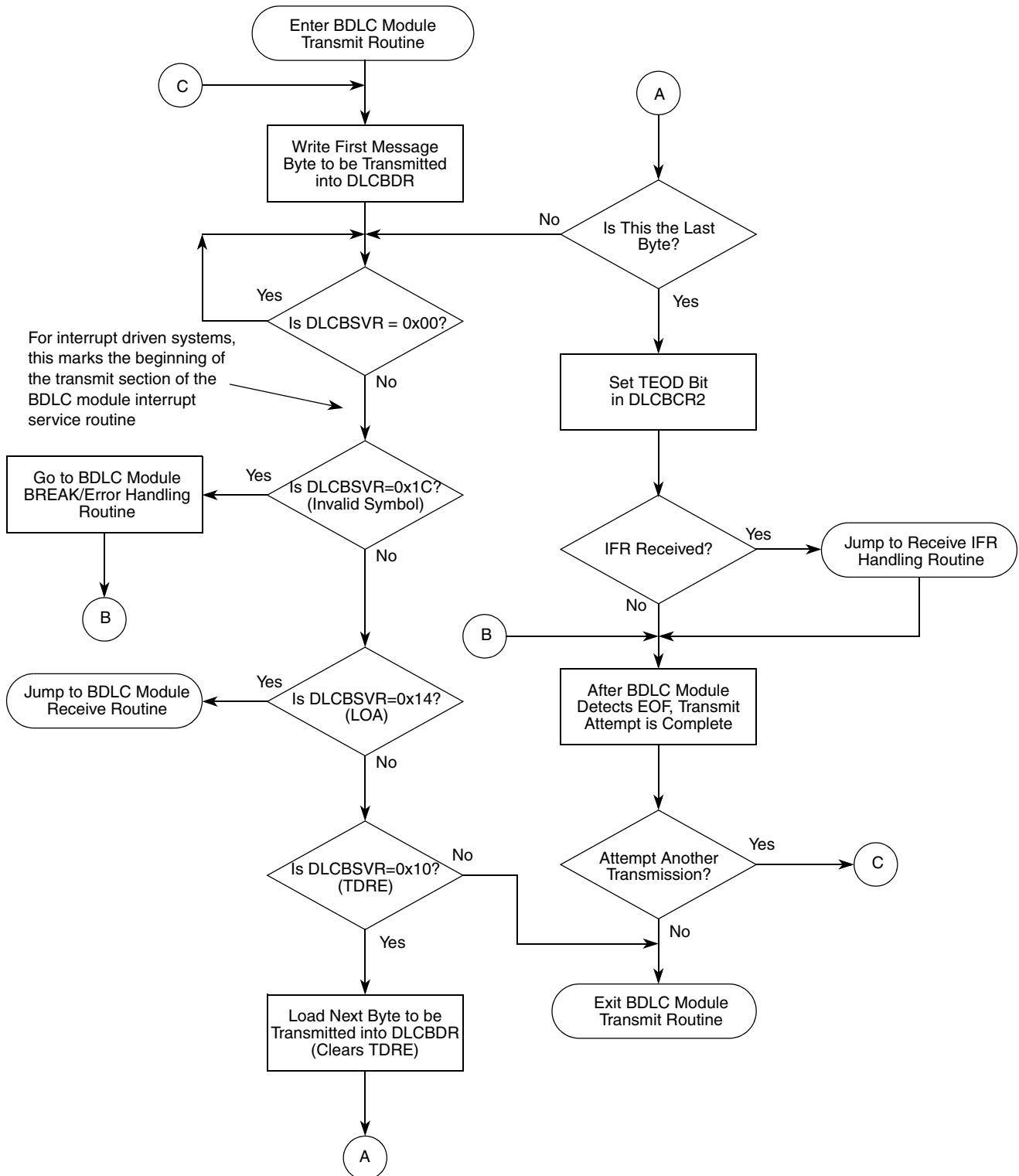
If a CRC error occurs following a transmission, this is also reflected in the BDLC State Vector Register. However, since the CRC error is really a receive error based on the received CRC byte, at this point all bytes of the message have been transmitted. It is therefore up to the user's software to determine if another attempt should be made to transmit the message in which the error occurred.

7.4.5.2.3 Transmitter Underrun

A transmitter underrun can occur when a TDRE interrupt is not serviced in a timely fashion. If the last byte loaded into the BDLC Data Register is completely transmitted onto the network before the next byte is loaded into the BDLC Data Register, a transmitter underrun occurs. If this does happen, the BDLC module transmits two additional logic ones to ensure that the partial message which was transmitted onto the bus does not end on a byte boundary. This is followed by an EOD and EOF symbol. The only indication to the CPU that an underrun occurred is the Symbol Invalid or Out of Range error indicated in the BDLC State Vector Register. As with the other errors, it is up to the user's software to determine if another transmission attempt should be made.

7.4.5.2.4 In-Frame Response to a Transmitted Message

If an In-Frame Response (IFR) is received following the transmission of a message, the status indicating that an IFR byte has been received is indicated in the BDLC State Vector Register before an EOF is indicated. Refer to [Section 7.4.8, "Receiving An In-Frame Response \(IFR\)"](#) for a description of how to manage the reception of IFR bytes.



NOTE: The EOF and CRC error interrupts are handled in the BDLC Module Receive Routine

Figure 7-20. Basic BDLC Transmit Flowchart

7.4.5.3 Aborting a Transmission

The BDLC module does not have a mechanism designed specifically for aborting a transmission. Because the module transmits each message on a byte-by-byte basis, there is little need to implement an abort mechanism. If the user has loaded a byte into the BDLC Data Register to initiate a message transmission and decides to send a different message, the byte in the BDLC Data Register can be replaced, right up to the point that the message transmission begins.

If the user has loaded a byte into the BDLC Data Register and then decides not to send any message at all, the user can let the byte transmit, and when the TDRE interrupt occurs let the transmitter underrun. This causes two extra logic ones followed by an EOF to be transmitted. While this method may require a small amount of bus bandwidth, the need to do this should be rare. Replacing the byte originally written to the BDLC Data Register with 0xFF also increases the probability of the transmitter losing arbitration if another node begins transmitting at the same time, also reducing the bus bandwidth needed.

7.4.6 Receiving A Message

The design of the BDLC module makes it especially easy to use for receiving messages off of the SAE J1850 bus. When the first byte of a message comes in, the BDLC State Vector Register indicates to the CPU that a byte has been received. As each successive byte is received, it is in turn be reflected in the BDLC State Vector Register. When the message is complete and the EOF has been detected on the bus, the BDLC State Vector Register reflects this, indicating that the message is complete.

The basic steps required for receiving a message from the SAE J1850 bus are outlined below. For more information on receiving IFR bytes, refer to [Section 7.4.8, “Receiving An In-Frame Response \(IFR\)”](#).

7.4.6.1 BDLC Reception Control Bits

The only control bit which is used for message reception, the IMMSG bit, is actually used to prevent message reception. When the IMMSG bit is set BDLC module interrupts of the CPU are inhibited until the next SOF symbol is received. This allows the BDLC module to ignore the remainder of a message once the CPU has determined that it is of no interest. This helps reduce the amount of CPU overhead used to service messages received from the SAE J1850 network, since otherwise the BDLC module would require attention from the CPU for each byte broadcast on the network. The IMMSG bit is cleared when the BDLC module receives an SOF symbol, or it can also be cleared by the CPU.

NOTE

While the IMMSG bit can be used to prevent the CPU from having to service the BDLC module for every byte transmitted on the SAE J1850 bus, the IMMSG bit should never be used to ignore the BDLC module's own transmission. Because setting the IMMSG bit prevents all BDLC State Vector Register bits from being updated until an SOF is received, the user would not receive any further transmit-related interrupts until another SOF was received, making it difficult for the CPU to complete the transmission correctly.

7.4.6.2 Receiving a Message with the BDLC Module

Receiving a message using the BDLC module is extremely straight-forward. As each byte of a message is received and placed into the BDLC Data Register, the BDLC module indicates this to the CPU with an Rx Data Register Full (RDRF) status in the BDLC State Vector Register. When an EOF symbol is received, indicating to the CPU that the message is complete, this is reflected in the BDLC State Vector Register.

Outlined below are the basic steps to be followed for receiving a message from the SAE J1850 bus with the BDLC module. For an illustration of this sequence, refer to [Figure 7-21](#).

1. When RDRF Interrupt Occurs, Retrieve Data Byte

When the first byte of a message following a valid SOF symbol is received that byte is placed in the BDLC Data Register, and an RDRF state is reflected in the BDLC State Vector Register. No indication of the SOF reception is made, since the end of the previous message is marked by an EOF indication. The first RDRF state following this EOF indication should allow the user to determine when a new message begins.

The RDRF interrupt is cleared when the received byte is read from the BDLC Data Register. After this is done, no further CPU intervention is necessary until the next byte is received, and this step is repeated.

All bytes of the message, including the CRC byte, are placed into the BDLC Data Register as they are received for the CPU to retrieve.

2. When an EOF is Received, the Message is Complete

After all bytes (including the CRC byte) have been received from the bus, the bus is idle for a time period equal to an EOD symbol. After the EOD symbol is received, the BDLC module verifies that the CRC byte is correct. If the CRC byte is not correct, this is reflected in the BDLC State Vector Register.

If no In-Frame Response bytes are transmitted following the EOD symbol, the EOD transitions into an EOF symbol. When the EOF is received it is reflected in the BDLC State Vector Register, indicating to the user that the message is complete. If IFR bytes do follow the first EOD symbol, once they are complete another EOD is transmitted, followed by an EOF.

After the EOF state is reflected in the BDLC State Vector Register, this indicates to the user that the message is complete, and that when another byte is received it is the first byte of a new message.

7.4.6.3 Filtering Received Messages

No message filtering hardware is included on the BDLC module, so all message filtering functions must be performed in software. Because the BDLC module handles each message on a byte-by-byte basis, message filtering can be done as each byte is received, rather than after the entire message is complete. This enables the CPU to decide while a message remains in progress whether or not that message is of any interest.

At any point during a message, if the CPU determines that the message is of no interest the IMMSG bit can be set. Setting the IMMSG bit commands the BDLC module not to update the BDLC State Vector Register until the next valid SOF is received. This prevents the CPU from having to service the BDLC module for each byte of every message sent over the network.

7.4.6.4 Receiving Exceptions

As with a message transmission, this basic message reception flow can be interrupted if errors are detected by the BDLC module. This can occur if an incorrect CRC is detected or if an invalid or out of range symbol appears on the SAE J1850 bus. A problem can also arise if the CPU fails to service the BDLC Data Register in a timely manner during a message reception.

7.4.6.4.1 Receiver Overrun

After a message byte has been received, the CPU must service the BDLC Data Register before the next byte is received, or the first byte is lost. If the BDLC Data Register is not serviced quickly enough, the next byte received is written over the previous byte in the BDLC Data Register. No receiver overrun indication is made to the CPU. If the CPU fails to service the BDLC module during the reception of an entire message, the byte remaining in the BDLC Data Register is the last byte received (usually a CRC byte).

After a receiver overrun occurs, there is no way for the CPU to recover the lost byte(s), so the entire message should be discarded. To prevent receiver overrun, the user should ensure that a BDLC RDRF interrupt is serviced before the next byte can be received. When polling the BDLC State Vector Register, the user should select a polling interval that provides timely monitoring of the BDLC module.

7.4.6.4.2 CRC Error

If a CRC error is detected during a message reception, this is reflected in the BDLC State Vector Register once an EOD time is recognized by the BDLC module. Because all bytes of the message have been received when this error is detected, it is up to the user to ensure that all the received message bytes are discarded.

7.4.6.4.3 Invalid or Out of Range Symbol

If an invalid or out of range symbol, a framing error or a BREAK symbol is detected on the SAE J1850 bus during the reception of a message, the BDLC module immediately stops receiving the message and discard any partially received byte. The symbol invalid or out of range status is immediately reflected in the BDLC State Vector Register. Following this, the BDLC module waits until the bus has been idle for a time period equal to an EOF symbol before receiving another message. As with the CRC error, the user should discard any partially received message if this occurs.

7.4.6.4.4 In-Frame Response to a Received Message

As mentioned above, if one or more IFR bytes are received following the reception of a message, the status indicating the reception of the IFR byte(s) is indicated in the BDLC State Vector Register before the EOF is indicated. Refer to [Section 7.4.8, “Receiving An In-Frame Response \(IFR\)”](#) for a description of how to deal with the reception of IFR bytes.

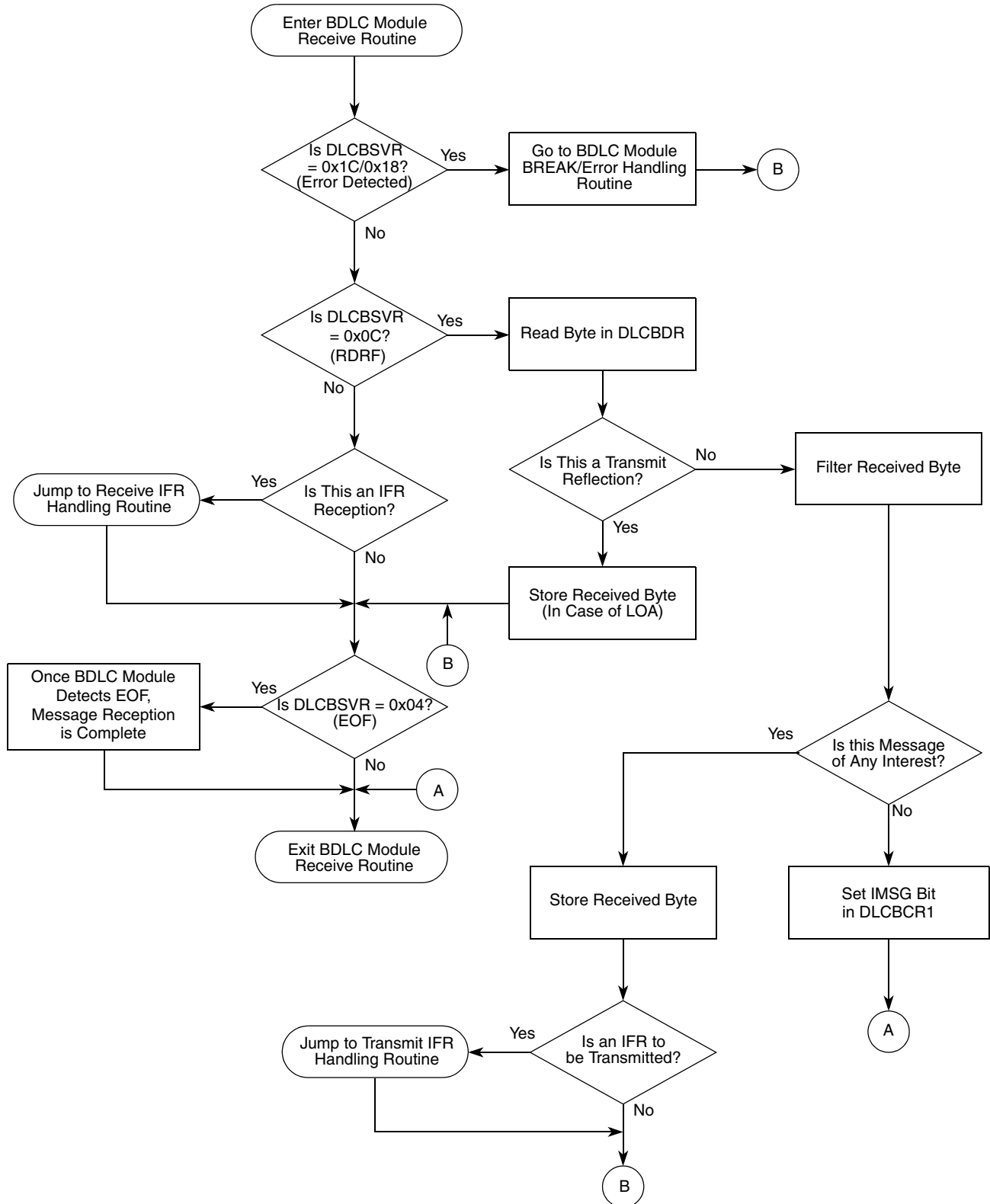


Figure 7-21. Basic BDLC Receive Flowchart

7.4.7 Transmitting an In-Frame Response (IFR)

The BDLC module can be used to transmit all four types of In-Frame Response (IFR) which are defined in SAE J1850. A brief definition of each IFR type is given below. For a more detailed description of each, refer the SAE J1850 document.

The explanation regarding IFR support by the BDLC module assumes familiarity with the use of IFRs as defined in SAE J1850 and understands the message header bit encoding and normalization bit formats used with the different types of IFRs. For more information on this, refer to the SAE J1850 document.

7.4.7.1 IFR Types Supported by the BDLC Module

SAE J1850 defines four distinct types of IFR. The first IFR is Type 0, or no IFR. IFR types 1, 2 and 3 are each made up of one or more bytes and, depending upon the type used, may be followed by a CRC byte. The BDLC module is designed to allow the user to transmit and receive all types of SAE J1850 IFRs, but only the network framing/error checking/bus acquisition duties are performed by the BDLC module. The user is responsible for determining the type of IFR to be transmitted, the number of retries to be made (if allowed), and the maximum number of bytes to be transmitted.

7.4.7.1.1 IFR Type 0: No Response

Generally, no IFR is used. The Type 0 IFR, as defined in SAE J1850, is no response. The EOD and EOF symbols follow directly after the CRC byte at the end of the message frame being transmitted. This type of IFR is inherently supported by the BDLC module, with no additional user intervention required.

7.4.7.1.2 IFR Type 1: Single Byte from a Single Responder

SAE J1850 defines the Type 1 IFR as a single byte from a single receiver. This type of IFR is used to acknowledge to the transmitter that the message frame was transmitted successfully on the network, and that at least one receiver received it correctly. A Type 1 IFR generally consists of the physical node ID of the receiver responding to the message, with no CRC byte appended. This type of response is used for broadcast-type messages, where there may be several intended receivers for a message, but the transmitter only wants to know that at least one node received it. In this case, all receivers begin transmitting their node ID following the EOD. Because all nodes on an SAE J1850 network have a unique node ID, if multiple nodes begin transmitting their node ID simultaneously, arbitration takes place. The node with the highest priority (lowest value) ID wins this arbitration process, and that node's ID makes up the IFR. No retries are attempted by the nodes which lose arbitration during a Type 1 IFR transmission.

A Type 1 IFR can also be used as a response to a physically addressed message, where the only intended receiver is the one which responds. In this case, no arbitration would take place during the IFR transmission, but the resulting IFR would consist of a single byte.

7.4.7.1.3 IFR Type 2: Single Byte from Multiple Responders

The Type 2 IFR, as defined in SAE J1850, is a series of single bytes, each transmitted by a different responder. This IFR type not only acknowledges to the transmitter that the message was transmitted successfully, but also reveals which receivers actually received the message. As with the Type 1 IFR, no CRC byte is appended to the end of a Type 2 IFR.

This IFR type is typically used with Function-type messages, where the original transmitter may need to know which nodes actually received the message. The basic difference between this type of IFR and the Type 1 IFR is that the nodes which lose arbitration while attempting to transmit their node ID during a Type 2 IFR wait until the byte which wins arbitration is transmitted and then again attempt to transmit their node ID onto the bus. The result is a series of node IDs, one from each receiver of the original message.

7.4.7.1.4 IFR Type 3: Multiple Bytes from a Single Responder

The last type of IFR defined by SAE J1850 is the Type 3 IFR. This IFR type consists of one or more bytes from a single responder. This type of IFR is used to return data to the original transmitter within the original message frame. This type of IFR may or may not have a CRC byte appended to it.

The Type 3 IFR is typically used with Function Read-type or Function Query-type messages, where the original transmitter is requesting data from the intended receiver. The node requesting the data transmits the initial portion of the message, and the intended receiver responds by transmitting the desired data in an IFR. In most cases, the original message requiring a Type 3 IFR is addressed to one particular node, so no arbitration should take place during the IFR portion of the message.

7.4.7.2 BDLC IFR Transmit Control Bits

The BDLC module has three bits which are used to control the transmission of an In-Frame Response. These bits, all located in BDLC Control Register 2, are TSIFR, TMIFR1 and TMIFR0. Each is used in conjunction with the TEOD bit to transmit one of three IFR types defined in SAE J1850. What follows is a brief description of each bit.

Because each of the bits used for transmitting an IFR with the BDLC module is used to transmit a particular type of IFR, only one bit should be set by the CPU at a time. However, should more than one of these bits get set at one time, a priority encoding scheme is used to determine which type of IFR is sent. This scheme prevents unpredictable operation caused by conflicting signals to the BDLC module.

[Table 7-22](#) illustrates which IFR bit is acted upon by the BDLC module should multiple IFR bits get set at the same time.

NOTE

As with transmitted messages, IFRs transmitted by the BDLC module are also received by the BDLC module. For a description of how IFR bytes received by the BDLC module should be handled, refer to [Section 7.4.8](#), “Receiving An In-Frame Response (IFR)”.

Table 7-22. IFR Control Bit Priority Encoding

READ/WRITE			ACTUAL		
TSIFR	TMIFR1	TMIFR0	TSIFR	TMIFR1	TMIFR0
0	0	0	0	0	0
1	—	—	1	0	0
0	1	—	0	1	0
0	0	1	0	0	1

7.4.7.3 Transmit Single Byte IFR

The Transmit Single Byte IFR (TSIFR) bit in BDLC Control Register 2 is used to transmit Type 1 and Type 2 IFRs onto the SAE J1850 bus. If this bit is set after a byte is loaded into the BDLC Data Register, the BDLC module attempts to send that byte, preceded by the appropriate Normalization Bit, as a single byte IFR without a CRC. If arbitration is lost, the BDLC module automatically attempts to transmit the byte again (without a Normalization Bit) as soon as the byte winning arbitration completes transmission. Attempts to transmit the byte continue until the byte is successfully transmitted, the TEOD bit is set by the user, or an error is detected on the bus.

The user must set the TSIFR bit before the EOD following the main part of the message frame is received, or no IFR transmit attempts are made for the current message. If another node does transmit an IFR to this message or a reception error occurs, the TSIFR bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

The TSIFR bit is automatically cleared after the EOD following one or more IFR bytes has been received or an error is detected on the bus.

7.4.7.4 Transmit Multi-Byte IFR 1

The Transmit Multi-Byte IFR 1 (TMIFR1) bit is used to transmit an SAE J1850 Type 3 IFR with a CRC byte appended. If this bit is set after the user has loaded the first byte of a multi-byte IFR into the BDLC Data Register, the BDLC module begins transmitting that byte, preceded by the appropriate Normalization Bit, onto the SAE J1850 bus. After this happens, a TDRE interrupt occurs, indicating to the user that the next IFR byte should be loaded into the BDLC Data Register. When the last byte to be transmitted is written to the BDLC Data Register, the user sets the TEOD bit. This causes a CRC byte and an EOD symbol to be transmitted following the last IFR byte.

As with the TSIFR bit, the TMIFR1 bit must be set before the EOD symbol is received or it remains cleared and no IFR transmit attempts are made. The TMIFR1 bit is cleared after the CRC byte and EOD are transmitted, if an error is detected on the bus, if a loss of arbitration occurs during the IFR transmission or if a transmitter underrun occurs when the user fails to service the TDRE interrupt in a timely manner. If a loss of arbitration occurs while the Type 3 IFR is being transmitted, transmission halts immediately and the loss of arbitration is indicated in the BDLC State Vector Register.

7.4.7.5 Transmit Multi-Byte IFR 0

The Transmit Multi-Byte IFR 0 (TMIFR0) bit is used to transmit an SAE J1850 Type 3 IFR without a CRC byte appended. If this bit is set after the user has loaded the first byte of a multi-byte IFR into the BDLC Data Register, the BDLC module begins transmitting that byte, preceded by the appropriate Normalization Bit, onto the SAE J1850 bus. After this happens, a TDRE interrupt occurs, indicating to the user that the next IFR byte should be loaded into the BDLC Data Register. When the last byte to be transmitted is written to the BDLC Data Register, the user sets the TEOD bit. This causes an EOD symbol to be transmitted following the last IFR byte.

As with the TSIFR and TMIFR1 bits, the TMIFR0 bit must be set before the EOD symbol is received or it remains cleared and no IFR transmit attempts are made. The TMIFR0 bit is cleared after the CRC byte and EOD are transmitted, if an error is detected on the bus, if a loss of arbitration occurs during the IFR

transmission or if a transmitter underrun occurs when the user fails to service the TDRE interrupt in a timely manner. If a loss of arbitration occurs while the Type 3 IFR is being transmitted, transmission halts immediately and the loss of arbitration is indicated in the BDLC State Vector Register.

NOTE

The TMIFR0 bit should not be used to transmit a Type 1 IFR. If a loss of arbitration occurs on the last bit of a byte being transmitted using the TMIFR0 bit, two extra logic ones are transmitted to ensure that the IFR does not end on a byte boundary. This can cause an error in a Type 1 IFR.

7.4.7.6 Transmitting An IFR with the BDLC module

While the design of the BDLC module makes the transmission of each type of IFR similar, the steps necessary for sending each is discussed. Again, a discussion of the bytes making up any particular IFR is not within the scope of this document. For a more detailed description of the use of IFRs on an SAE J1850 network, refer to the SAE J1850 document.

7.4.7.6.1 Transmitting a Type 1 IFR

To transmit a Type 1 IFR, the user loads the byte to be transmitted into the BDLC Data Register and sets both the TSIFR bit and the TEOD bit. This directs the BDLC module to attempt transmitting the byte written to the BDLC Data Register one time, preceded by the appropriate Normalization Bit. If the transmission is not successful, the byte is discarded and no further transmission attempts are made. For an illustration of the steps described below, refer to [Figure 7-22](#).

1. Load the IFR Byte into the BDLC Data Register

The user begins initiation of a Type 1 IFR by loading the desired IFR byte into the BDLC Data Register. If a byte has already been written into the BDLC Data Register for transmission as a new message, the user can simply write the IFR byte to the BDLC Data Register, replacing the previously written byte. This must be done before the first EOD symbol is received.

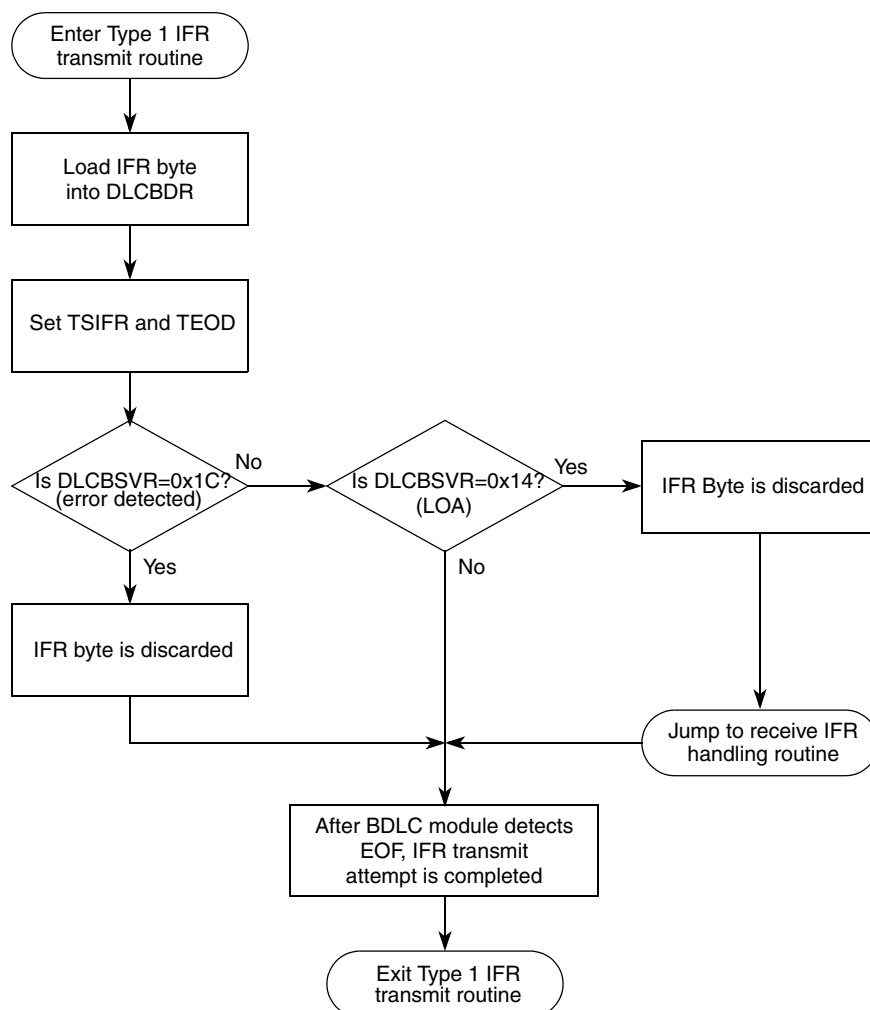


Figure 7-22. Transmitting A Type 1 IFR

2. Set the TSIFR and TEOD Bits

The final step in transmitting a Type 1 IFR with the BDLC module is to set the TSIFR and TEOD bits in BDLC Control Register 2. Setting both bits directs the BDLC module to make one attempt at transmitting the byte in the BDLC Data Register as an IFR. If the byte is transmitted successfully, or if an error or loss of arbitration occurs, TEOD and TSIFR are cleared and no further transmit attempts are made.

7.4.7.6.2 Transmitting a Type 2 IFR

To transmit a Type 2 IFR, the user loads the byte to be transmitted into the BDLC Data Register and sets the TSIFR bit. After this is done, the BDLC module attempts to transmit the byte in the BDLC Data Register as a single byte IFR, preceded by the appropriate Normalization Bit. If the first BDLC module loses arbitration on the first attempt, it makes repeated attempts to transmit this byte until it is successful,

an error occurs or the user sets the TEOD bit. For an illustration of the steps described below, refer to [Figure 7-23](#).

1. Load the IFR Byte into the BDLC Data Register

As with the Type 1 IFR, the user begins initiation of a Type 2 IFR by loading the desired IFR byte into the BDLC Data Register. If a byte has already been written into the BDLC Data Register for transmission as a new message, the user can simply write the IFR byte to the BDLC Data Register, replacing the previously written byte. This must be done before the first EOD symbol is received.

2. Set the TSIFR Bit

The second step necessary for transmitting a Type 2 IFR is to set the TSIFR bit in BDLC Control Register 2. Setting this bit directs the BDLC module to attempt to transmit the byte in the BDLC Data Register as an IFR until it is successful. If the byte is transmitted successfully or if an error or loss of arbitration occurs, TSIFR is cleared and no further transmit attempts are made.

3. If Necessary, Set the TEOD Bit

The third step in transmitting a Type 2 IFR is only necessary if the user wishes to halt the transmission attempts. This may be necessary if the BDLC module's attempt to transmit the byte loaded into the BDLC Data Register continually loses arbitration, and the overall message length approaches the 12-byte limit as defined in SAE J1850.

If it becomes necessary to halt the IFR transmission attempts, the user simply sets the TEOD bit in BDLC Control Register 2. If the BDLC module is between transmission attempts, it makes one more attempt to transmit the IFR byte. If it is transmitting the byte when TEOD is set, the BDLC module continues the transmission until it is successful or it loses arbitration to another transmitter. At this point, it then discards the byte and make no more transmit attempts.

NOTE

When transmitting a Type 2 IFR, the user should monitor the number of IFR bytes received to ensure that the overall message length does not exceed the 12-byte limit for the length of SAE J1850 messages. The user should set the TEOD bit when the 11th byte is received, which prevents the 12-byte limit from being exceeded.

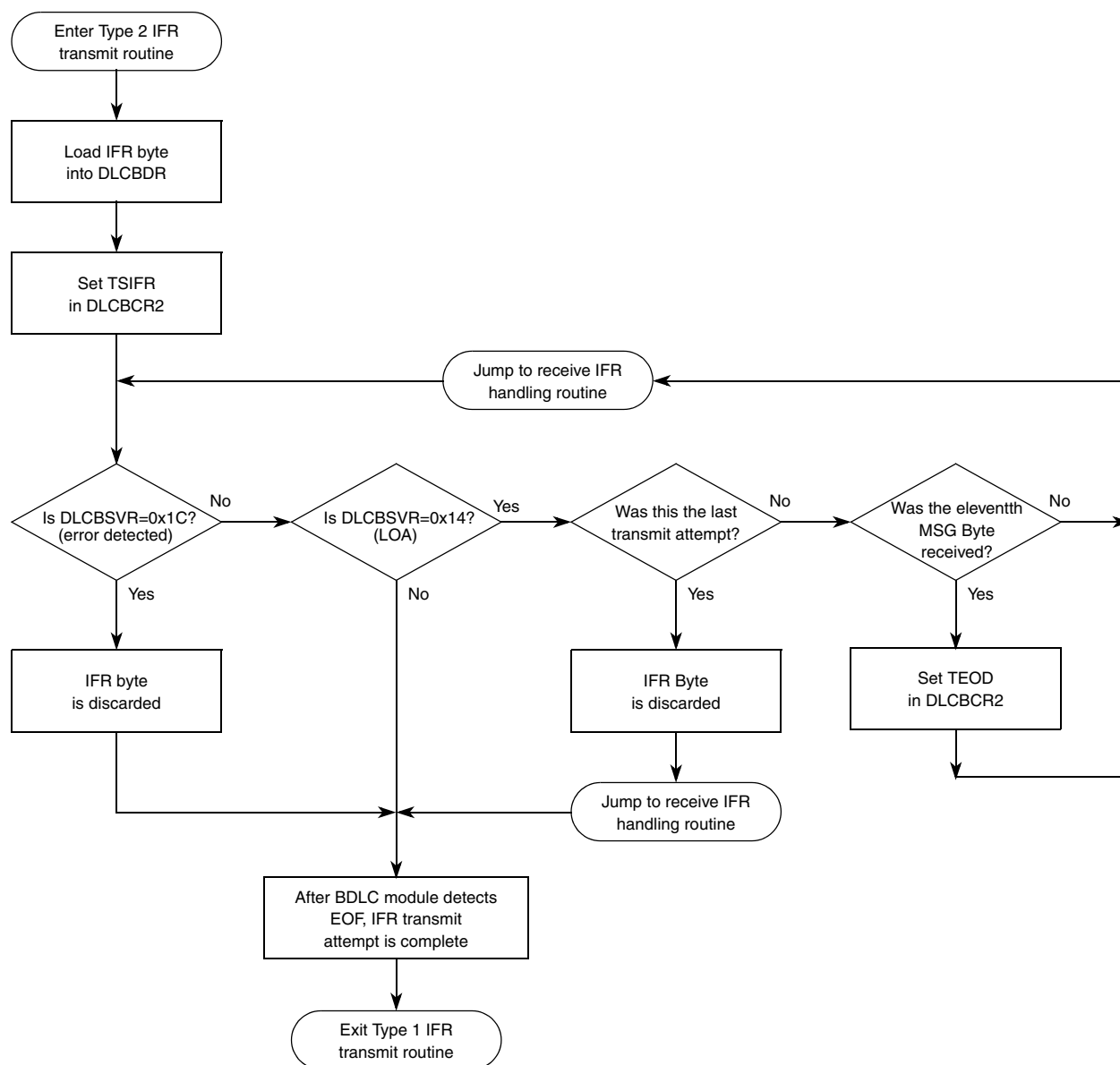


Figure 7-23. Transmitting A Type 2 IFR

7.4.7.6.3 Transmitting a Type 3 IFR

Transmitting a Type 3 IFR, with or without a CRC byte, is done in a fashion similar to transmitting a message frame. The user loads the first byte to be transmitted into the BDLC Data Register and then sets the appropriate TMIFR bit, depending upon whether a CRC byte is desired. When the last byte is written to the BDLC Data Register, the TEOD bit is set, and a CRC byte (if desired) and an EOD are then transmitted. Because the two versions of the Type 3 IFR are transmitted identically, the description that follows discusses both. For an illustration of the Type 3 IFR transmit sequence, refer to [Figure 7-24](#).

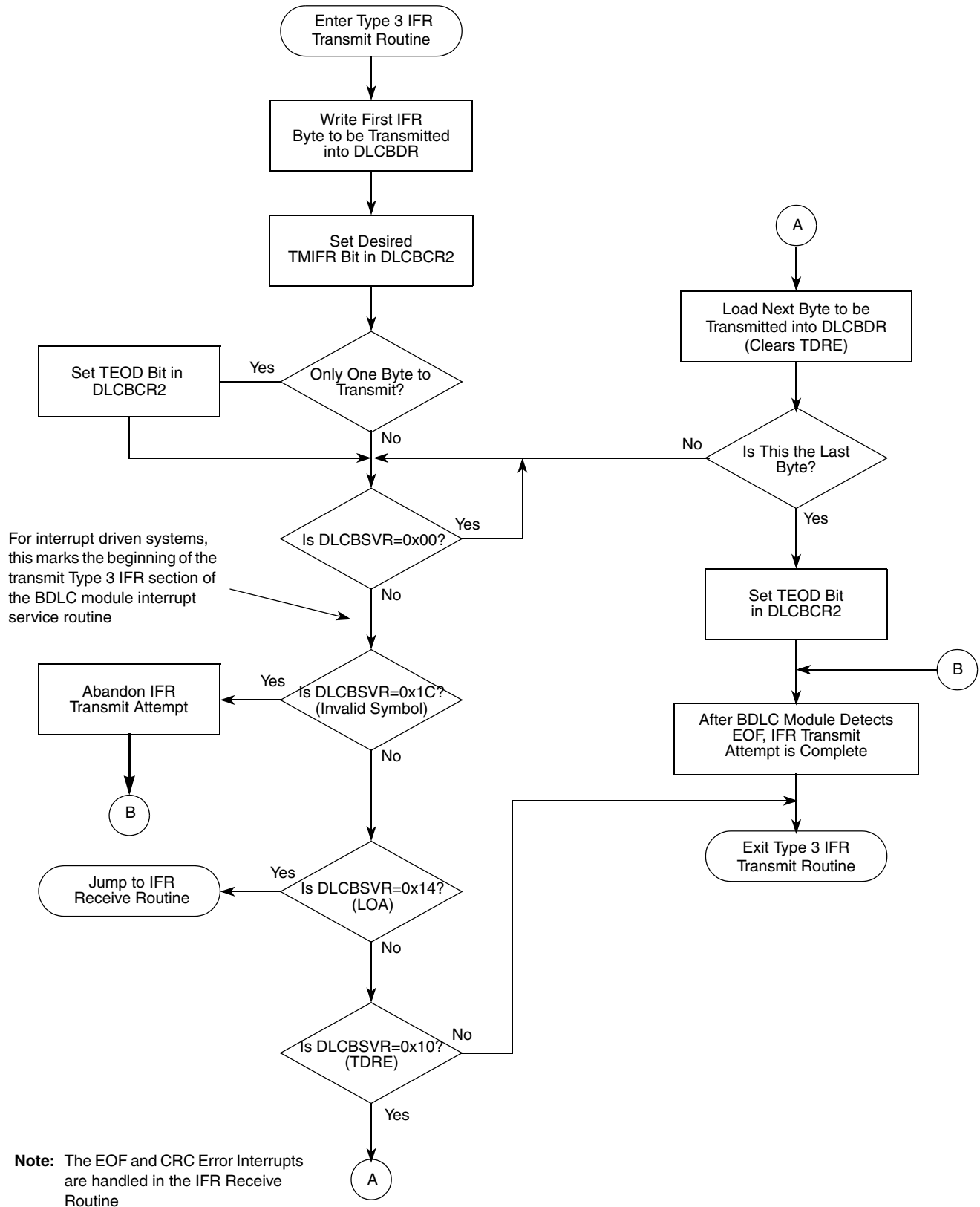


Figure 7-24. Transmitting A Type 3 IFR

1. Load the First IFR Byte into the BDLC Data Register

The user begins initiation of a Type 3 IFR, as with each of the other IFR types, by loading the desired IFR byte into the BDLC Data Register. If a byte has already been written into the BDLC Data Register for transmission as a new message, the user can simply write the first IFR byte to the BDLC Data Register, replacing the previously written byte. This must be done before the first EOD symbol is received.

2. Set the TMIFR Bit

The second step necessary for transmitting a Type 3 IFR is to set the desired TMIFR bit in BDLC Control Register 2, depending upon whether or not a CRC is desired. As previously described in [Section 7.4.7.2, “BDLC IFR Transmit Control Bits”](#), the TMIFR1 bit should be set if the user requires a CRC byte to be appended following the last byte of the Type 3 IFR, and TMIFR0 if no CRC byte is required.

Setting the TMIFR1 or TMIFR0 bit directs the BDLC module to transmit the byte in the BDLC Data Register as the first byte of a single or multi-byte IFR preceded by the appropriate Normalization Bit. After this has occurred, the BDLC State Vector Register reflects that the next byte of the IFR can be written to the BDLC Data Register (TDRE interrupt).

NOTE

The user must set the TMIFR1 or TMIFR0 bit before the EOD following the main part of the message frame is received or no IFR transmit attempts are made for the current message. If another node does transmit an IFR to this message or a reception error occurs, the TMIFR1 or TMIFR0 bit is cleared. If not, the IFR is transmitted after the EOD of the next received message.

3. When TDRE is Indicated, Write the Next IFR Byte into the BDLC Data Register

When a TDRE state is reflected in the BDLC State Vector Register, the CPU writes the next IFR byte to be transmitted into the BDLC Data Register, clearing the TDRE interrupt. This step is repeated until the last IFR byte to be transmitted is written to the BDLC Data Register.

NOTE

When transmitting a Type 3 IFR, you may write two or three of the bytes to be transmitted into the BDLC Data Register before the first Rx IFR interrupt occurs. For this reason, never use receive IFR byte interrupts to control the sequencing of IFR bytes to be transmitted.

4. Write the Last IFR Byte into the BDLC Data Register and Set TEOD

After the last IFR byte to be transmitted is written to the BDLC Data Register, the CPU then sets the TEOD bit in BDLC Control Register 2. After the TEOD bit is set and the last IFR byte written to the BDLC Data Register is transmitted onto the bus (if the TMIFR1 bit has been set), the BDLC module begins transmitting the CRC byte, followed by an EOD. If the TMIFR0 bit has been set, the last IFR byte is immediately followed by the transmission of an EOD. Following the EOD, EOF is recognized and the message is complete.

If a loss of arbitration occurs at any time during the transmission of a Type 3 IFR, the TMIFR bit is set, and the TEOD bit (if set) is cleared, any IFR byte being transmitted is discarded and the loss of arbitration state is reflected in the BDLC State Vector Register. Likewise, if an error is detected

during the transmission of a Type 3 IFR the IFR control bits are cleared, the byte being transmitted is discarded, and the BDLC State Vector Register reflects the detected error.

If the Type 3 IFR being transmitted is made up of a single byte, the appropriate TMIFR bit and the TEOD bit can be set at the same time. The BDLC module then treats that byte as the first and last IFR byte to be sent.

7.4.7.7 Transmitting IFR Exceptions

This basic IFR transmitting flow can be interrupted for the same reasons as a normal message transmission. The IFR transmit process can be adversely affected due to a loss of arbitration, an Invalid or Out of Range Symbol, or due to a transmitter underrun caused by the CPU failing to service a TDRE interrupt in a timely fashion. For a description of how these exceptions can affect the IFR transmit process, refer to [Section 7.4.5.2, “Transmitting Exceptions”](#).

7.4.8 Receiving An In-Frame Response (IFR)

Receiving an In-Frame Response with the BDLC module is similar to receiving a message frame. As each byte of an IFR is received, the BDLC State Vector Register indicates this to the CPU. An EOF indication in the BDLC State Vector Register indicates that the IFR (and message) is complete. Also, the IMMSG bit can also be used to command the BDLC module to mask any further network activity from the CPU, including IFR bytes being received, until the next valid SOF is received.

NOTE

As with a message transmission, the IMMSG bit should never be used to ignore the BDLC module's own IFR transmissions. This is again due to the BDLC State Vector Register bits being inhibited from updating until IMMSG is cleared, preventing the CPU from detecting any IFR-related state changes which may be of interest.

7.4.8.1 Receiving an IFR with the BDLC Module

Receiving an IFR from the SAE J1850 bus requires the same procedure that receiving a message does, except that as each byte is received the Received IFR Byte (RxIFR) state is indicated in the BDLC State Vector Register. All other actions are the same. For an illustration of the steps described below, refer to [Figure 7-25](#).

1. When RxIFR Interrupt Occurs, Retrieve IFR Byte

When the first byte of an IFR following a valid EOD symbol is received that byte is placed in the BDLC Data Register, and an RxIFR state is reflected in the BDLC State Vector Register. No indication of the EOD reception is made because the RxIFR state indicates that the main portion of the message has ended and the IFR portion has begun.

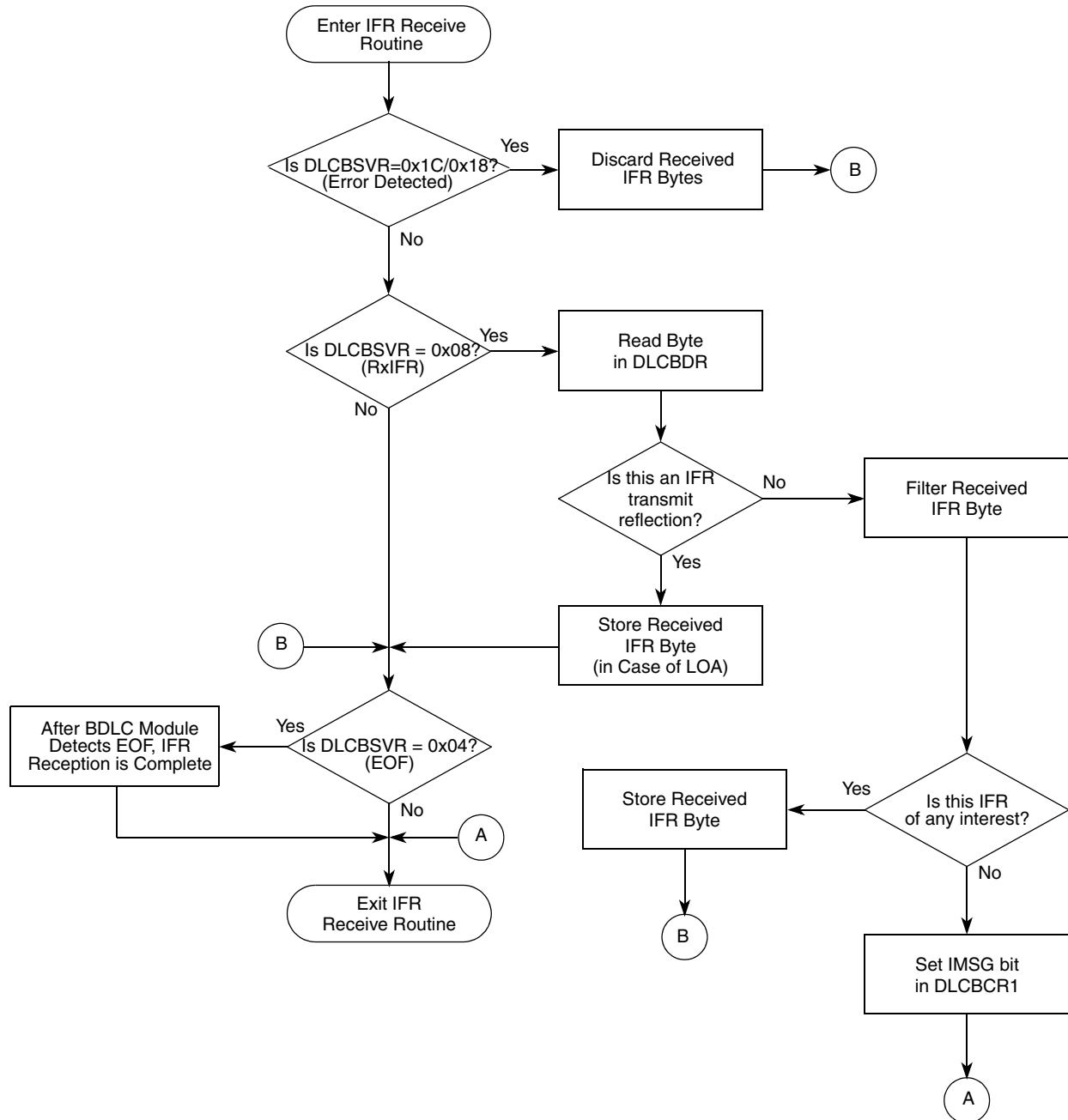


Figure 7-25. Receiving An IFR with the BDLC Module

The RxIFR interrupt is cleared when the received IFR byte is read from the BDLC Data Register. After this is done, no further CPU intervention is necessary until the next IFR byte is received, and this step is repeated. As with a message reception, all bytes of the IFR, including the CRC byte, are placed into the BDLC Data Register as they are received for the CPU to retrieve.

2. When an EOF is Received, the IFR (and Message) is Complete

After all IFR bytes (including the possible CRC byte) have been received from the bus, the bus is idle again for a time period equal to an EOD symbol. Following this, the BDLC module determines

whether or not the last byte of the IFR is a CRC byte, and if so verify that the CRC byte is correct. If the CRC byte is not correct, this is reflected in the BDLC State Vector Register.

After an additional period of time, the EOD symbol transitions into an EOF symbol. When the EOF is received, it is reflected in the BDLC State Vector Register, indicating to the user that the IFR and the message is complete.

7.4.8.2 Receiving IFR Exceptions

This basic IFR receiving flow can be interrupted for the same reasons as a normal message reception. The IFR receiving process can be adversely affected due to a CRC error, an Invalid or Out of Range Symbol or due to a receiver overrun caused by the CPU failing to service an RxIFR interrupt in a timely fashion. For a description of how these exceptions can affect the IFR receiving process, refer to [Section 7.4.6.4, “Receiving Exceptions”](#).

7.4.9 Special BDLC Module Operations

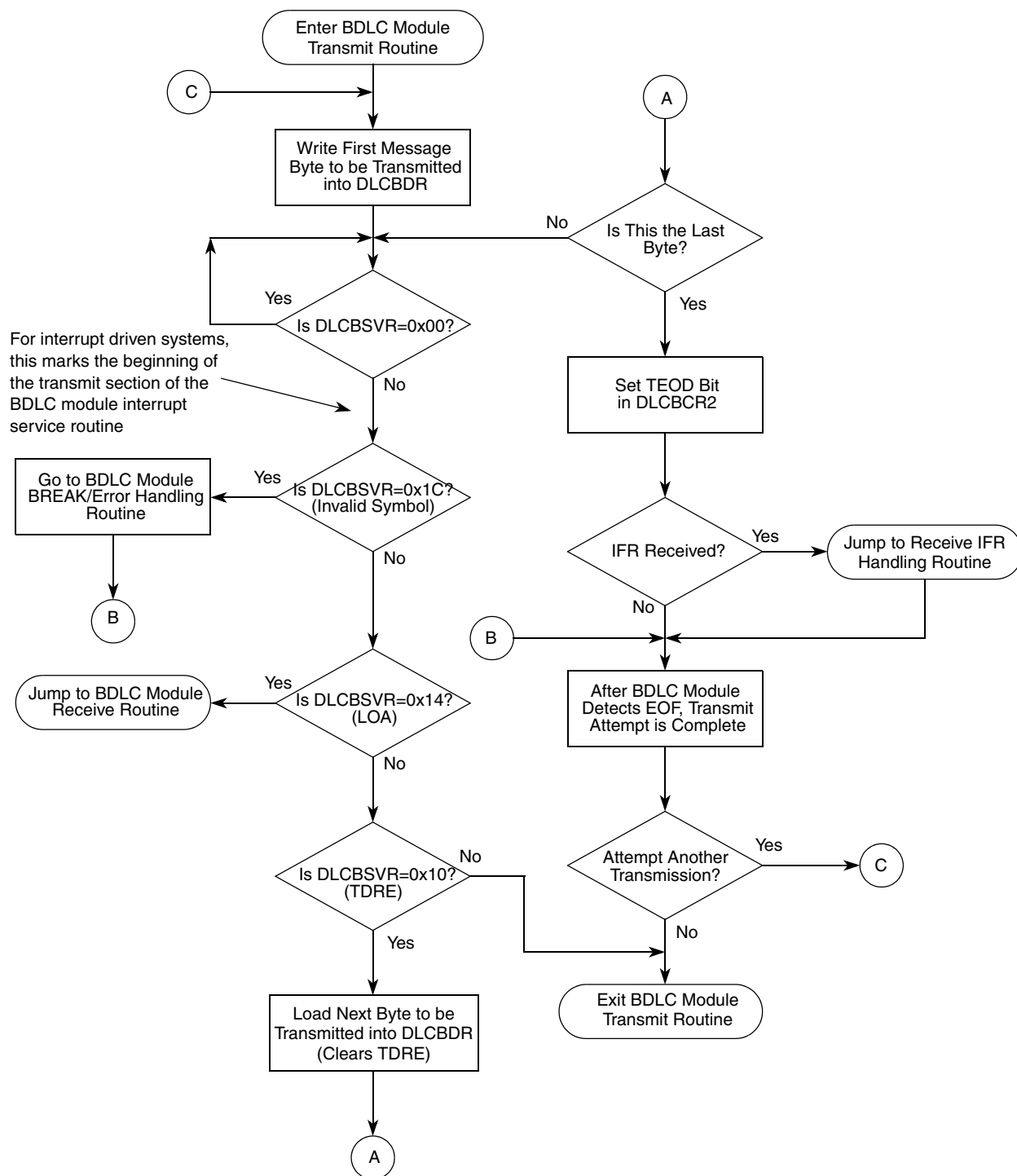
There are a few special operations which the BDLC module can perform. What follows is a brief description of each of these functions and when they might be used.

7.4.9.1 Transmitting Or Receiving A Block Mode Message

The BDLC module, because it handles each message on a byte-by-byte basis, has the inherent capability of handling messages any number of bytes in length. While during normal operation this requires the user to carefully monitor message lengths to ensure compliance with SAE J1850 message limits, often in a production or diagnostic environment messages which exceed the SAE J1850 limits can be beneficial. This is especially true when large amounts of configuration data need to be downloaded over the SAE J1850 network.

Because of the BDLC module's architecture, it can both transmit and receive messages of unlimited length. The CRC calculations for transmitting and receiving are not limited to eight bytes, but are instead calculated and verified using all bytes in the message, regardless of the number. All control bits, including TEOD and MSG, also work in an identical manner, regardless of the length of the message.

To transmit or receive these block mode messages, no extra BDLC module control functions must be performed. The user simply transmits or receives as many bytes as desired in one message frame, and the BDLC module operates as if a message of normal length was being used.



NOTE: The EOF and CRC Error Interrupts are handled in the BDLC Module Receive Routine

Figure 7-26. Basic BDLC Module Transmit Flowchart

7.4.9.2 Transmitting Or Receiving A Message In 4X Mode

In a diagnostic or production environment large amounts of data may need to be downloaded across the network to a component or module. This data is often sent in a large block mode message (see above) which violates the SAE J1850 limit for message length. To speed up the downloading of these large blocks of data, they are sometimes transmitted at four times (4X) the normal bit rate for the Variable Pulse Width modulation version of SAE J1850. This higher speed transmission, nominally 41.6kbps, allows these large blocks to be transmitted much more quickly.

The BDLC module is designed to receive and transmit messages at this higher speed. By setting the 4XE bit in BDLC Control Register 2, the user can command the BDLC module to transmit and receive any message over the network at a 4X rate.

If the BDLC module is placed in this 4X mode, messages transmitted at the normal bit rate are not received correctly. Likewise, 4X messages transmitted on the SAE J1850 bus when the BDLC module is in normal mode are interpreted as noise on the network by the BDLC module. For more information on the 4XE bit, refer to [4X Mode](#).

7.5 Initialization Information

To initialize the BDLC module, the user should first write the desired data to the configuration bits. The BDLC module should then be taken out of digital and analog loopback mode and enabled. Exiting from loopback mode entails change of state indications in the BDLC State Vector Register which must be dealt with. After this is complete, CPU interrupts can be enabled (if desired), and then the BDLC module is capable of SAE J1850 serial network communication. For an illustration of the sequence necessary for initializing the BDLC module, refer to [Figure 7-27](#).

7.5.1 Initializing the Configuration Bits

The first step necessary for initializing the BDLC module following an MCU reset is to write the desired values to each of the BDLC module control registers. This is best done by storing predetermined initialization values directly into these registers. The following description outlines a basic flow for initializing the BDLC module. This basic flow does not detail more elaborate initialization routines, such as performing digital and analog loopback tests before enabling the BDLC module for SAE J1850 communication. However, from the following descriptions and the BDLC module specification, the user should be able to develop routines for performing various diagnostic procedures such as loopback tests.

1. Initialize BDLC Analog Round Trip Delay Register

Begin initialization of the configuration bits by writing the desired analog transceiver configuration data into the BDLC Analog Round Trip Delay Register. Following this write to BDLC Analog Round Trip Delay Register, all of these bits become read only.

2. Initialize BDLC Baud Rate Select Register

The next step in BDLC module initialization is to write the desired bus clock divisor minus one into the BDLC Baud Rate Select Register. The divisor should be chosen to generate a 1 MHz or 1.048576 MHz mux interface clock (f_{bdlc}). Following this write to BDLC Baud Rate Select Register, all of these bits become read only.

3. Initialize BDLC Control Register 2

The next step in BDLC module initialization should be writing the configuration bits into the BDLC Control Register 2 register. This initialization description assumes the BDLC module is put into normal mode (not 4X mode), and that the BDLC module should not yet exit digital or analog loopback mode. Therefore, this step should write SMRST and DLOOP as logic ones, 4XE as a logic zero, write NBFS to the desired level, and write TEOD, TSIFR, TMIFR1 and TMIFR0 as logic zeros. These last four bits MUST be written as logic zeros to prevent undesired operation of the BDLC module.

4. Initialize BDLC Control Register 1

The next step in BDLC module initialization is to write the configuration bits in BDLC Control Register 1. The CLKS bit should be written to its desired values at this time, following which it becomes read-only. The IE bit should be written as a logic zero at this time so BDLC module interrupts of the CPU remain masked for the time being. The IMSG bit should be written as a logic one to prevent any receive events from setting the BDLC State Vector Register until a valid SOF (or BREAK) symbol has been received by the BDLC module.

7.5.2 Exiting Loopback Mode and Enabling the BDLC Module

After the configuration bits have been written to the desired values, the BDLC module should be taken out of loopback and connected to the SAE J1850 bus. This is done by clearing the DLOOP bit and then setting the BDLCE bit in the BDLC Control Register.

1. Perform Loopback Tests (optional)

After the BDLC module is configured for desired operation, the user may wish to perform digital and/or analog loopback tests to determine the integrity of the link to the SAE J1850 network. This would involve leaving the DLOOP bit (BDLC Control Register 2) set, setting the BDLCE bit, performing the desired loopback tests and finally exiting digital loopback mode by clearing DLOOP in the BDLC Control Register 2.

2. Exit Loopback Mode and Enable the BDLC Module

If loopback mode tests are not to be performed the BDLC module can be removed from digital loopback mode by clearing the DLOOP bit. The BDLC module can then be enabled by setting the BDLCE bit in the BDLC Control Register.

After DLOOP is cleared and BDLCE is set, the BDLC module is ready for SAE J1850 communication. However, to ensure that the BDLC module does not attempt to receive a message already in progress or to transmit a message while another device is transmitting, the BDLC module must first observe an EOF symbol on the bus before the receiver is activated. To activate the transmitter, the BDLC module needs to observe an Inter-Frame Separator symbol.

7.5.3 Enabling BDLC Interrupts

The final step in readying the BDLC module for proper communication is to clear any pending interrupt sources and then, if desired, enable BDLC module interrupts of the CPU.

1. Clear Pending BDLC Interrupts

To ensure that the BDLC module does not immediately generate a CPU interrupt when interrupts are enabled, the user should read the BDLC State Vector Register to determine if any BDLC module interrupt sources are pending before setting the IE bit in the BDLC Control Register 1. If the BDLC State Vector Register reads as a %00000000, no interrupts are pending and the user is free to enable BDLC interrupts, if desired.

If the BDLC State Vector Register indicates that an interrupt is pending, the user should perform whatever actions are necessary to clear the interrupt source before enabling the interrupts. Whether any interrupts are pending depends primarily upon how much time passes between the exit from loopback modes and enabling the BDLC module and the enabling of interrupts. It is a good practice to always clear any source of interrupts before enabling interrupts on any MCU subsystem.

If any interrupts are pending (BDLC State Vector Register not %00000000), then each interrupt source should be dealt with accordingly. After all of the interrupt sources have been dealt with, the BDLC State Vector Register should read %00000000, and the user is then free to enable BDLC interrupts.

2. Enable BDLC Interrupts

The last step in initializing the BDLC module is to enable interrupts to the CPU, if so desired. This is done by simply setting the IE bit in the BDLC Control Register 1. Following this, the BDLC module is ready for operating in interrupt mode. If the user chooses not to enable interrupts, the BDLC State Vector Register must be polled periodically to ensure that state changes in the BDLC module are detected and dealt with appropriately.

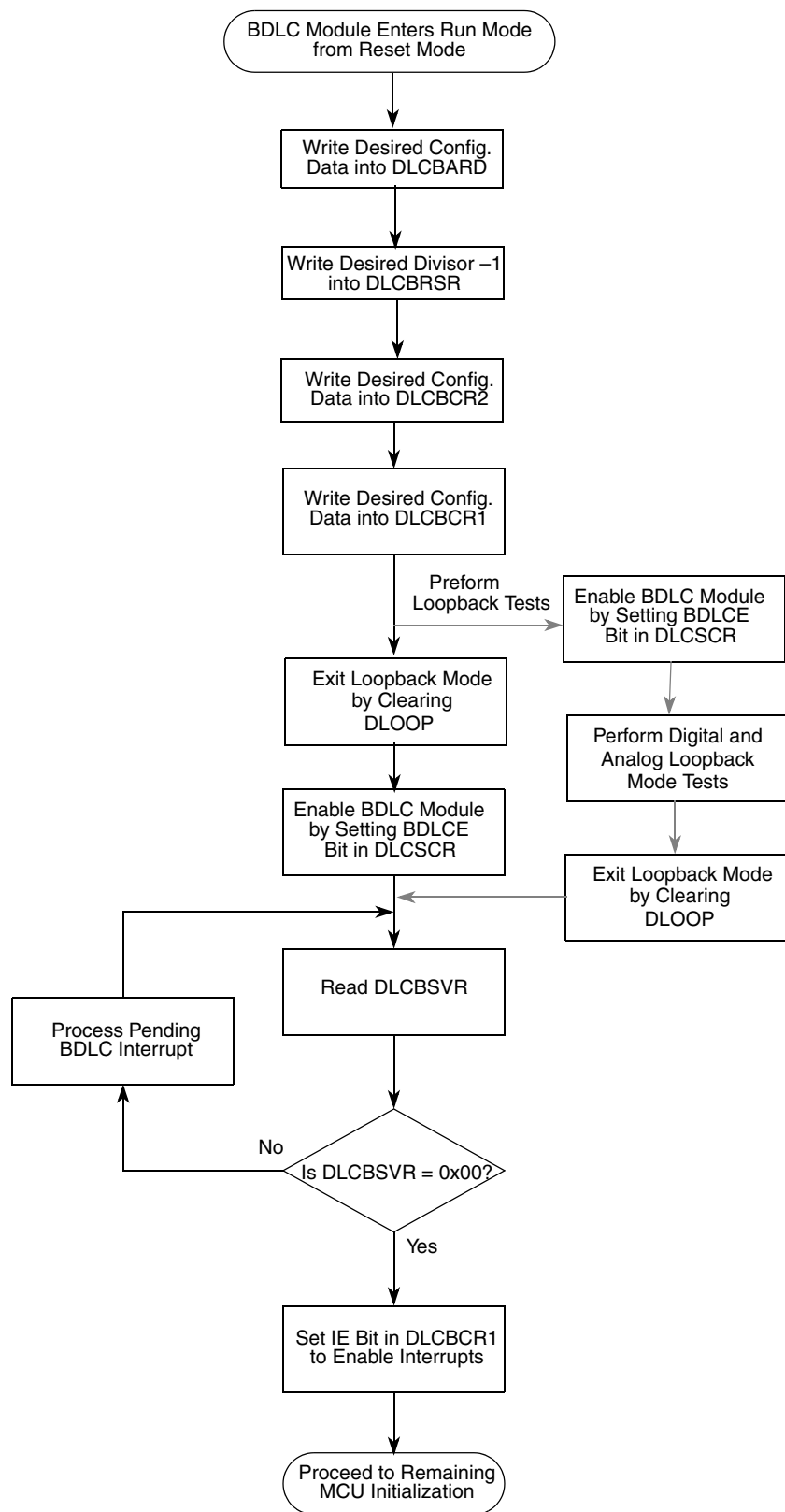


Figure 7-27. Basic BDLC Module initialization Flowchart

Chapter 8

Clock Frequency Measurement (CFM)

8.1 Introduction

8.1.1 Overview

The CFM, clock frequency measurement, is the module used to measure the frequency of the muxed input clocks of MPC5121e in digital way. The module measures the frequency of input clock as a function of the frequency of IPG_CLK and stores the result in the register. Figure 8-1 displays the block diagram of CFM. The FreqMeas Circuit can calculate the relation value between the system clock(IPG_CLK) and the clocks to be measured (Input_clk shown in the figure, be selected from one of CLK_IN1-CLK_IN4). The measure result is stored in the FreqMeas Register. The Config Register controls the clock selection logic and provides parameter for FreqMeas Circuit. All registers can be accessed via IPS bus.

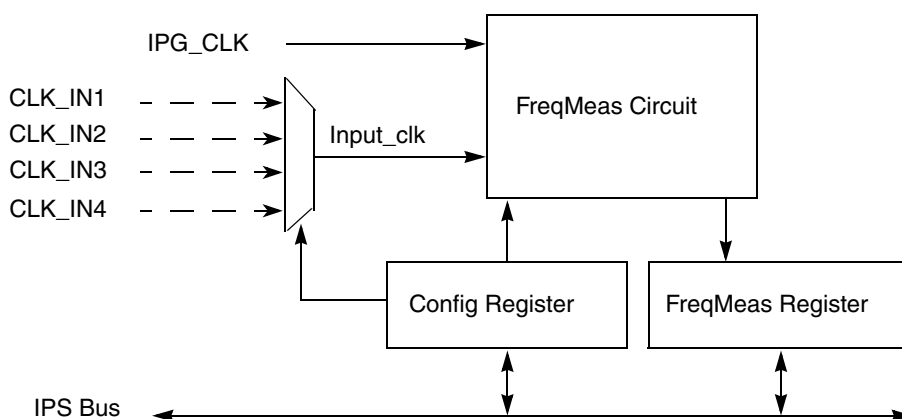


Figure 8-1. Diagram of the CFM

8.1.2 Features

- 4-Channel frequency measurement of externally received slave clocks
- Support a sufficient measurement of audio waveform frequency range (64*Audio_Sample, The Audio_Sample ranged in 32 kHz, 44.1 kHz, 48 kHz, 64 kHz, 88.2 kHz, and 96 kHz)
- Accuracy of measurement: the result with no more than 1e-4 relative error can be get after 1e7 IPG_CLK cycles measurement time.
- The gain¹ value configurable to adjust the measurement result to fill a 32-bit register

1. The gain value can be used to adjust the result of frequency measure. The frequency measure result is a ratio between the frequency of measured clock and IPG_CLK and is stored in a 32-bit register. A suitable gain value can make the result close to a 32-bit value.

8.2 Memory Map and Register Definition

8.2.1 Memory Map

[Table 8-1](#) displays the CFM memory-mapped 32-bit registers, described in Register Descriptions. The bit 0 is MSB, bit 31 is LSB.

Table 8-1. CFM Block Memory Map

Address CFM_BAS+	Register	Access	Section/Page
0x00	PhaseConfig (FPC)—Frequency measure configuration register	R/W	8.2.2.1/8-3
0x08	FreqMeas (FMS)—Frequency measure result register	R	8.2.2.2/8-4

8.2.2 Register Descriptions

8.2.2.1 PhaseConfig Register

The PhaseConfig is R/W register, which includes the information of coef(gain) selection and clock source selection for frequency measurement as follows.

Offset: CFM_BAS + 0x00 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	CoefSel		ClkSrc_Sel			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 8-2. PhaseConfig Register

Table 8-2. PhaseConfig Field Descriptions

Field	Description
CoefSel	<p>This is the gain value for measurement result. To make the measue result more close to a 32-bit value. For instance, a smaller value could be selected responding to a lower frequency of measued clock.</p> <p>COEF selection:</p> <p>000 24 001 16 010 12 011 8 100 6 101 4 Others: 3</p>
ClkSrc_Sel	<p>Clock source selection:</p> <p>000 CLK_IN1 001 CLK_IN2 010 CLK_IN3 011 CLK_IN4 Others: Reserved</p> <p>Totally 4 clock sources can be selected to be measured: CLK_IN1-CLK_IN4(reference to Figure 8-1). All of them come from clock module.</p> <p>For the detail of these 4 clock sources, please reference 6.4.1.20: CFM Clock Control Register (CCCR) register description in Clocks and Low-Power Modes Chapter.</p>

8.2.2.2 FreqMeas Register

This register is used to save the result of frequency measurement of the input clock source. It is read only.

Offset: CFM_BAS + 0x08Access: User read only

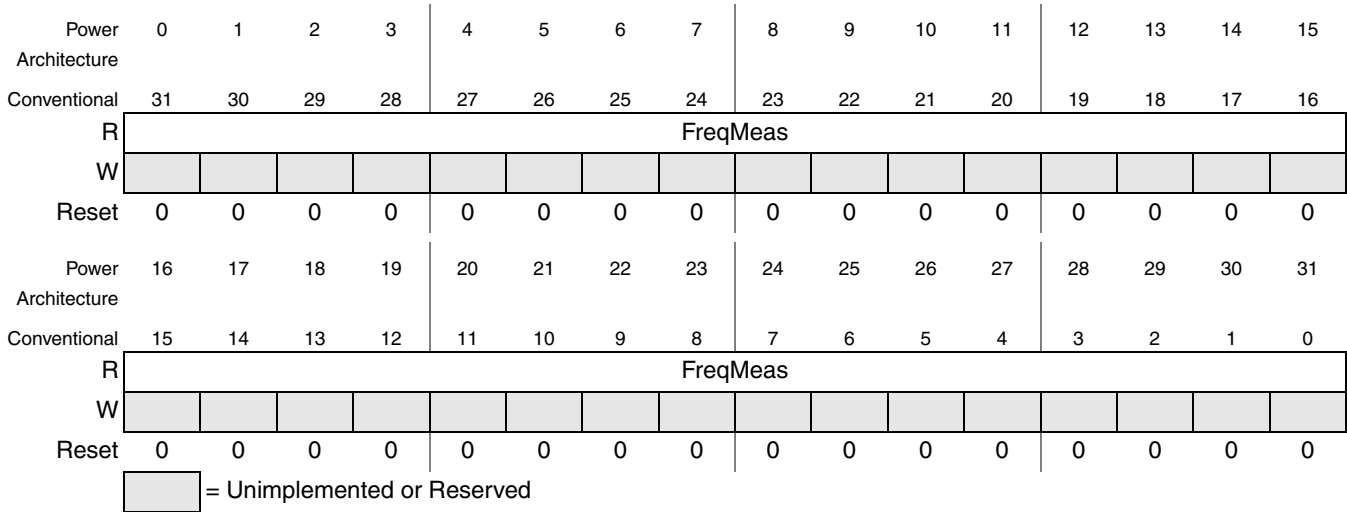


Figure 8-3. FreqMeas Register

Table 8-3. FreqMeas Field Descriptions

Field	Description
FreqMeas	The result of frequency measurement. It's a frequency ratio between measured clock and IPG_CLK: Freqmeas_CLK/IPG_CLK * (2**28) * COEF

8.3 Functional Description

The diagram of frequency measurement is shown in [Figure 8-1](#).

Associated with it, are 2 registers, PhaseConfig and FreqMeas. The GAIN (Coef) is programmable by CPU writing the CoefSel bits field of the register PhaseConfig. This register is also responsible for the testing clock selection, CLK_IN1~CLK_IN4. The register FreqMeas is used to save the measurement result in 32-bit and it can be read by CPU through bus.

The circuit measures the frequency of the incoming clock Freqmeas_CLK(muxed from clk_in1~clk_in4) as a function of the IPG_CLK, as follows.

$$\text{Freqmeas_CLK} = \text{FreqMeas}[31:0] * \text{IPG_CLK} * /((2^{**}28) * \text{COEF}) \text{ --- (1)}$$

The circuit is a second-order filter. The output is a value represented by an unsigned number stored in the 32-bit FreqMeas register, giving the ratio of testing clock frequency vs bus clock frequency. In the circuit, the frequency measure value shrinks to certain values after enough waiting period. Therefore, after configuration, it needs a certain period (generally 6 million IPG_CLK cycles) before reading the FreqMeas register to get an accurate result. A more accurate result is generated with more time.

8.4 Application Example

Here is an example for application. If the CLK_IN2 need to be FreqMeased, and COEF is selected as 8, the steps are as following.

1. Write the register PhaseConfig as 32'h0000_0019.
2. Wait for 1e7 IPG_CLK cycles, then read the register FreqMeas to get the result no more than 100ppm(ppm means 1e-6) relative error.
3. Use formula (1) to get the actual frequency of CLK_IN2 .

If another clock need to be measured, repeat step 1 – 3.

Chapter 9

CPU e300 Core Power Architecture

9.1 Introduction

The following sections are contained in this document:

- [e300c4lp Processor Core Functional Overview](#)
- [e300c4lp Core Reference Manual](#)
- [Unsupported e300c4lp Core Features](#)

9.2 e300c4lp Processor Core Functional Overview

The module integrates a e300c4lp processor core based on, and compatible with, the 603e which is a Power Architecture compliant microprocessor. The e300c4lp core is completely embedded, as its address, data, and control signals are not externally visible. The e300c4lp core has the following features:

- e300 Power Architecture Core
- Dual Issue, superscalar architecture
- 32-Kbyte instruction cache, 32K data cache
- Double precision FPU
- Instruction and data MMU
- Power management modes:
 - Nap
 - Doze
 - Sleep
- Standard and critical interrupt capability

For additional information on the capabilities and features of the e300c4lp core, refer to e300 user documentation.

After power-on or hard reset, initial boot instructions are fetched from the LocalPlus bus, with CS0 active, or from page 0 of NAND flash. To facilitate high speed execution, boot code is typically copied from a flash or ROM device to SDRAM. The e300c4lp core can execute code from the on-chip SRAM.

The e300c4lp core has memory mapped access to all resources, including:

- All on-chip programming registers
- External SDRAM
- Internal SRAM

- PCI-controlled address space
- External disk drive control register space (via PIO mode), etc.

Bursting is supported on the CS Bus. Critical word first protocol is employed when the e300c4lp core attempts to fill its address and data caches.

The ID values for the MPC5121e are shown in Table 9-1.

Table 9-1. ID Values for the MPC5121e

	M36P Mask Set
PVR	0x8086_2010
MPC5121e SVR	0x8018_0020
MPC5123 SVR	0x8018_0030
PCI Device ID	0x580C
PCI Vendor ID	0x1957
JTAG ID Code	0x1540_a01d

9.3 e300c4lp Core Reference Manual

A complete specification for the e300c4lp core implementation used on the module is obtained through a collection of documentation.

- Power Architecture Microprocessor Family: The Programming Environments for 32-bit Microprocessors, Rev. 2: MPCFPE32B/AD
- e300 Power Architecture Core Family Reference Manual, Rev. 4

The programming environments manual provides information about resources defined by the Power Architecture architecture common to Power Architecture processors. Implementation variances relative to Rev. 2 of the Programming Environments Manual are available in the e300 Core Reference Manual.

The e300 Power Architecture Core Family Reference Manual can be obtained from the Freescale Literature Distribution center at <http://www.freescale.com>.

9.4 Unsupported e300c4lp Core Features

9.4.1 Instructions

Two Power Architecture instructions are not supported by the module. These two instructions are `eciwx` and `ecowx`. The execution of both instructions generates a $\overline{\text{TEA}}$ signal on CSB. This causes a machine check exception or a checkstop.

9.4.2 CSB Parity

Enabling of the address or data parity error check by setting the `HID0[EBA, EBD]` bits generates a machine check exception or a checkstop depending on the `HID0[EMCP]` bit.

9.4.3 Performance Monitor Event

The Performance Monitor Event input (e300_pm_event_in) is deactivated. It is tied to 0.

Chapter 10

CSB Arbiter and Bus Monitor

10.1 Introduction

This chapter describes operation theory of the CSB arbiter in the MPC5121e device. In addition, it describes configuration, control and status registers of the arbiter.

The CSB arbiter is responsible for providing coherent system bus arbitration. It tracks all the address and data tenures, and provides all the arbitration signals to masters and slaves. In addition, it monitors the bus and reports on errors and protocol violations.

10.1.1 Features

The CSB arbiter includes the following features:

- Supports a programmable pipeline depth (from 1 to 4)
- Supports four levels of priority for bus arbitration
- Supports repeat request mode: number of programmable consecutive transactions from the same master (up to eight transactions)
- Supports data streaming operations
- Supports programmable address bus parking mode: disable, park to last bus owner, park to s/w selected master
- Claims address only, reserved, and illegal transaction types, report on it and can raise maskable interrupt
- Provides timers for address tenure time-out and data tenure time-out detection and can issue maskable interrupt, if any timer expired
- Reports on transfer error and can issue maskable interrupt
- Can issue regular or machine check interrupt for each type of error event (programmable)

10.1.1.1 Coherent System Bus Overview

Coherent system bus is the central bus. Any data transaction from master to slave in the device passes through the coherent system bus. The coherent system bus supports pipelined transactions. It has independent address and data tenures. Pipeline depth determines the number of address tenures that can be started before the first data tenure is finished.

Basic burst size is equal to cache line length of Power Architecture core, which is 32 bytes. Using repeat request mode enables up to eight consecutive bursts to be executed by the same master. Maximum number of consecutive transactions can be limited by programming arbiter configuration register (See [Section 10.2.1.1, “Arbiter Configuration Register \(ACR\)”](#) for more details).

10.2 Memory Map/Register Definition

[Table 10-1](#) shows the memory map for arbiter’s configuration, control and status registers.

Table 10-1. Arbiter Register Map

Memory Offset (Hex)	Register	Access	Section/Page
0x00	ACR—Arbiter Configuration Register	4-byte R/W	10.2.1.1/10-3
0x04	ATR—Arbiter Timers Register	4-byte R/W	10.2.1.2/10-5
0x08	ATER—Arbiter Transfer Error Register	4-byte R/W	10.2.1.3/10-6
0x0c	AER—Arbiter Event Register	4-byte R/W	10.2.1.4/10-7
0x10	AIDR—Arbiter Interrupt Definition Register	4-byte R/W	10.2.1.5/10-9
0x14	AMR—Arbiter Mask Register	4-byte R/W	10.2.1.6/10-10
0x18	AEATR—Arbiter Event Attributes Register	4-byte R	10.2.1.7/10-12
0x1c	AEADR—Arbiter Event Address Register	4-byte R	10.2.1.8/10-15
0x20	AERR—Arbiter Event Response Register	4-byte R/W	10.2.1.9/10-16

10.2.1 Register Descriptions

10.2.1.1 Arbiter Configuration Register (ACR)

Arbiter configuration register (ACR) defines the arbiter modes and parked master on the bus. Figure 10-1 shows the fields of ACR.

Offset 0x00 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	DTO_DIS	ATO_DIS	CORE_DIS	0	0	AACKWS		0	PIPE_DEP		
W																
Reset	0	0	0	0	0	0	0	0 ¹	0	0	0 ¹	0 ¹	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	PCI_RPTCNT			0	RPTCNT			0	W_PARK	APARK		PARKM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. The reset value of COREDIS and AACKWS is determined from reset configuration word.

= Unimplemented or Reserved

Figure 10-1. Arbiter Configuration Register (ACR)
(The register is repeated for reference.)

Table 10-2. ACR Field Descriptions

Field	Description
DTO_DIS	Data time out detection disable 1 Stop the DTO counter and prevent data time out detection
ATO_DIS	Address time out detection disable 1 Stop the ATO counter and prevent address time out detection
COREDIS	This bits needs always be set to zero.
AACKWS	Reserved. Write should preserve reset value. Address acknowledge wait states. Specifies minimum number of address tenure wait states. This is the minimum delay between assertion of \overline{TS} and assertion of \overline{AACK} . 00 \overline{AACK} is asserted minimum 1 cycle after \overline{TS} 01 \overline{AACK} is asserted minimum 2 cycles after \overline{TS} 10 \overline{AACK} is asserted minimum 3 cycles after \overline{TS} 11 \overline{AACK} is asserted minimum 4 cycles after \overline{TS} Note: 00 option can be used only if the processor is not operating in 1:1 or 3:2 bus clock ratios

Table 10-2. ACR Field Descriptions (continued)

Field	Description
PIPE_DEP	<p>Pipeline depth (number of outstanding transactions).</p> <p>000 Pipeline depth 1 (1 outstanding transaction)</p> <p>001 Pipeline depth 2 (2 outstanding transactions)</p> <p>010 Pipeline depth 3 (3 outstanding transactions)</p> <p>011 Pipeline depth 4 (4 outstanding transactions)</p> <p>100 Reserved (Pipeline depth 5)</p> <p>101 Reserved (Pipeline depth 6)</p> <p>110 Reserved (Pipeline depth 7)</p> <p>111 Reserved (Pipeline depth 8)</p>
PCI_RPTCNT	<p>PCI/PCI DMA repeat count. Specifies the maximum number of consecutive transactions, that PCI/PCI DMA master can perform, using $\overline{\text{REPEAT}}$ request mode.</p> <p>000 1 consecutive transaction ($\overline{\text{REPEAT}}$ request mode disable)</p> <p>001 2 consecutive transactions</p> <p>010 3 consecutive transactions</p> <p>011 4 consecutive transactions</p> <p>100 5 consecutive transactions</p> <p>101 6 consecutive transactions</p> <p>110 7 consecutive transactions</p> <p>111 8 consecutive transactions</p>
RPTCNT	<p>Repeat count. Specifies the maximum number of consecutive transactions, that any master (except PCI/PCI DMA) can perform, using $\overline{\text{REPEAT}}$ request mode.</p> <p>000 1 consecutive transactions ($\overline{\text{REPEAT}}$ request mode disable)</p> <p>001 2 consecutive transactions</p> <p>010 3 consecutive transactions</p> <p>011 4 consecutive transactions</p> <p>100 5 consecutive transactions</p> <p>101 6 consecutive transactions</p> <p>110 7 consecutive transactions</p> <p>111 8 consecutive transactions</p> <p>Note: It is recommended not to program this field for more than 4 consecutive transactions.</p>
WPARK	<p>WOP Parking. Specifies, whether bus is parked to CPU on WOP cycle (cycle after $\overline{\text{ARTRY}}$ assertion).</p> <p>0 Park to CPU</p> <p>1 Don't park bus to any master at WOP cycle</p>
APARK	<p>Address parking. Specifies arbiter bus parking mode.</p> <p>00 Park to master. Arbiter parks the address bus to the master, that is selected by numeric value of PARKM field.</p> <p>01 Park to last owner. Arbiter parks the address bus to last bus owner.</p> <p>10 Disable. Arbiter does not assert $\overline{\text{BG}}$ to any master, if no $\overline{\text{BR}}$ is present.</p> <p>11 Reserved</p>
PARKM	<p>Parking master.</p> <p>0000 Power Architecture Core</p> <p>0001 PCI/PCI DMA</p> <p>0010 TPR2MG/SAP</p> <p>0011 Reserved</p> <p>0100 Reserved</p> <p>0101 Reserved</p> <p>0110–1111 Reserved</p>

10.2.1.2 Arbiter Timers Register (ATR)

Arbiter timers register (ATR) defines the arbiter address time out (ATO), data time out (DTO) timer's values. Figure 10-2 shows the fields of ATR.

Offset 0x04 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DTO															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ATO															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 10-2. Arbiter Timers Register (ATR)

Table 10-3. ATR Field Descriptions

Field	Description
DTO	Data time out. Specifies the time-out period for the data tenure. The granularity of this field is 128 bus clocks. The maximum value is 8388480 coherent system bus clocks. Data time_out occurs, if the data tenure was not ended before the specified time-out period timer expires between the assertion of \overline{DBB} until the assertion of last \overline{TA} . When $DTO = n$, the timeout cycle is $n \times 128$. 0x0000 Reserved 0x0001 128 clock cycles 0x0002 256 clock cycles 0x0003 384 clock cycles ... 0xFFFF 8388480 clock cycles
ATO	Address time out. Specifies the time-out period for the address tenure. The granularity of this field is 128 bus clocks. Maximum value is 8388480 coherent system bus clocks. Address time-out occurs, if the address tenure was not ended before the specified time-out period timer expires between assertion of \overline{TS} signal until the assertion of \overline{AACK} signal. When $ATO = n$, the timeout cycle is $n \times 128$. 0x0000 Reserved 0x0001 128 clock cycles 0x0002 256 clock cycles 0x0003 384 clock cycles 0xFFFF 8388480 clock cycles

Figure 10-3.

10.2.1.3 Arbiter Transfer Error Register (ATER)

Arbiter transfer error register (ATER) specifies which kind of events are considered error events. If event is defined as non error event, it also won't be reported neither in event register nor in event attributes and address registers. For transfer types, that are not defined as error events, arbiter also does not end address/data tenures. [Figure 10-4](#) shows the fields of ATER.

Offset 0x08Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0						
W											ETEA	RES	ECW	AO	DTO	ATO
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 10-4. Arbiter Transfer Error Register (ATER)

Table 10-4. ATER Field Descriptions

Field	Description
ETEA	External \overline{TEA} . Specifies, whether assertion of \overline{TEA} signal by one of the slaves is reported in arbiter event registers. 0 Assertion of \overline{TEA} signal by one of the slaves isn't reported in arbiter event registers. 1 Assertion of \overline{TEA} signal by one of the slaves is reported in arbiter event registers.
RES	Reserved transfer type. Specifies, whether transaction with reserved transfer type is reported in arbiter event registers. 0 Reserved transaction isn't reported in arbiter event registers. 1 Reserved transaction is reported in arbiter event registers.
ECW	External Control Word transfer type. Specifies, whether transaction with external control word transfer type is reported in arbiter event registers. 0 External control word read/write transaction isn't reported in arbiter event registers. 1 External control word read/write transaction is reported in arbiter event registers.
AO	Address Only transfer type. Specifies, whether transaction with address only transfer type is reported in arbiter event registers. 0 Address only transaction isn't reported in arbiter event registers. 1 Address only transaction is reported in arbiter event registers.
DTO	Data Time Out. Specifies, whether data tenure time out is reported in arbiter event registers. 0 Data time out isn't reported in arbiter event registers. 1 Data time out is reported in arbiter event registers.
ATO	Address Time Out. Specifies, whether address tenure time out is reported in arbiter event registers. 0 Address time out isn't reported in arbiter event registers. 1 Address time out is reported in arbiter event registers.

10.2.1.4 Arbiter Event Register (AER)

The arbiter uses arbiter event register (AER) to report on erroneous transactions. This register is cleared by writing 1's. [Figure 10-5](#) shows the fields of AER.

Offset 0x0cAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	ETEA	RES	ECW	AO	DTO	ATO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-5. Arbiter Event Register (AER)

Table 10-5. AER Field Descriptions

Field	Description
ETEA	Transfer error. External \overline{TEA} . Reports on detection of transfer error assertion of \overline{TEA} signal by one of the slaves. 0 No transfer error detected \overline{TEA} signal is not asserted by one of the slaves. 1 Transfer error detected \overline{TEA} signal is asserted by one of the slaves.
RES	Reserved transfer type. Reports on transaction with reserved transfer type. See Section 10.3.2.5, “Reserved Transaction Type,” for more information. 0 No transaction with reserved transfer type occurred. 1 Transaction with reserved transfer type occurred.
ECW	External control word transfer type. Reports on transaction with external control word transfer type. See Section 10.3.2.6, “Illegal (ECIW/ECOW) Transaction Type,” for more information. 0 No transaction with external control word transfer type occurred. 1 Transaction with external control word transfer type occurred.
AO	Address Only transfer type. Reports on transaction with address only transfer type. See Section 10.3.2.4, “Address Only Transaction Type,” for more information. 0 No transaction with address only transfer type occurred. 1 Transaction with address only transfer type occurred.
DTO	Data time out. Reports on data tenure time out. 0 Data time out timer is not expired. 1 Data time out timer is expired.
ATO	Address time out. Reports on address tenure time out. 0 Address time out timer is not expired. 1 Address time out timer is expired.

10.2.1.5 Arbiter Interrupt Definition Register (AIDR)

Arbiter interrupt definition register (AIDR) determines the interrupt that responds to different error conditions. Setting a bit defines the corresponding interrupt as MCP interrupt; clearing a bit defines the corresponding interrupt as regular interrupt. Figure 10-6 shows the fields of AIDR.

Offset 0x10 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0						
W											ETEA	RES	ECW	AO	DTO	ATO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-6. Arbiter Interrupt Definition Register (AIDR)

Table 10-6. AIDR Field Descriptions

Field	Description
ETEA	Transfer error. External \overline{TEA} . Detection of transfer error assertion of \overline{TEA} signal by one of the slaves interrupt definition. 0 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves causes regular interrupt. 1 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves causes MCP interrupt.
RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes regular interrupt. 1 transaction with reserved transfer type causes MCP interrupt.
ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes regular interrupt. 1 Transaction with external control word transfer type causes MCP interrupt.
AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes regular interrupt. 1 Transaction with address only transfer type causes MCP interrupt.

Table 10-6. AIDR Field Descriptions (continued)

Field	Description
DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes regular interrupt. 1 Data tenure time out causes MCP interrupt.
ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes regular interrupt. 1 Address tenure time out causes MCP interrupt.

10.2.1.6 Arbiter Mask Register (AMR)

Arbiter mask register (AMR) is used to mask interrupts or reset requests. Setting a mask bit enables the corresponding interrupt or reset request; clearing a bit masks it. Regular interrupts, MCP interrupts and reset requests can be masked by AMR register. Figure 10-7 shows the fields of AMR.

Offset 0x14 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	ETEA	RES	ECW	AO	DTO	ATO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-7. Arbiter Mask Register (AMR)

Table 10-7. AMR Field Descriptions

Field	Description
ETEA	Transfer error. External \overline{TEA} . Detection of transfer error assertion of \overline{TEA} signal by one of the slaves interrupt mask bit. 0 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves interrupt disabled. 1 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves interrupt enabled.
RES	Reserved transfer type. Transaction with reserved transfer type interrupt mask bit. 0 Transaction with reserved transfer type interrupt disabled. 1 Transaction with reserved transfer type interrupt enabled.

Table 10-7. AMR Field Descriptions (continued)

Field	Description
ECW	External control word transfer type. Transaction with external control word transfer type interrupt mask bit. 0 Transaction with external control word transfer type interrupt disabled. 1 Transaction with external control word transfer type interrupt enabled.
AO	Address only transfer type. Transaction with address only transfer type interrupt mask bit. 0 Transaction with address only transfer type interrupt disabled. 1 Transaction with address only transfer type interrupt enabled.
DTO	Data time out. Data tenure time out interrupt mask bit. 0 Data tenure time out interrupt disabled. 1 Data tenure time out interrupt enabled.
ATO	Address time out. Address tenure time out interrupt mask bit. 0 Address tenure time out interrupt disabled. 1 Address tenure time out interrupt enabled.

10.2.1.7 Arbiter Event Attributes Register (AEATR)

Arbiter event attributes register (AEATR) reports the type of transaction that causes error, which is specified in the event register. See [Section 10.2.1.4, “Arbiter Event Register \(AER\),”](#) for more information. AEATR is cleared only by power-on reset. The attributes of the first error event are stored. As AEATR is not effected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the bus is stalled. [Figure 10-8](#) shows the fields of AEATR.

Offset 0x18 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	EVENT		0	0	0	MSTR_ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	TBST	TSIZE		0	0	0	TTYPE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-8. Arbiter Event Attributes Register (AEATR)

Table 10-8. AEATR Field Descriptions (Sheet 1 of 2)

Field	Description
EVENT	Event type. 000 Address time out 001 Data time out 010 Address only transfer type 011 External control word transfer type 100 Reserved transfer type 101 Transfer error External TEA 110 Reserved 111 Reserved
MSTR_ID	Master Id. 00000 Power Architecture core data transaction 00001 Reserved 00010 Power Architecture core instruction fetch 00011–01001 Reserved 01010 JTAG 01011 TPR 01011–01100 Reserved 01101 PCI 01110 Reserved 01111 PCI DMA 10000–11111 Reserved Note: Master Id reflects the source of transaction and is used for debug purpose.
TBST	Transfer burst. 0 Transfer size is up to 8 bytes. 1 Transfer size is greater than 8 bytes.

Table 10-8. AEATR Field Descriptions (Sheet 2 of 2)

Field	Description
TSIZE	<p>Transfer Size. Transfer size encoding depends on the value of $\overline{\text{TBST}}$.</p> <p>$\overline{\text{TBST}} = 1$:</p> <ul style="list-style-type: none"> 0011 Byte 0102 Bytes 0113 Bytes 1004 Bytes 1015 Bytes 1106 Bytes 1117 Bytes 0008 Bytes <p>$\overline{\text{TBST}} = 0$:</p> <ul style="list-style-type: none"> 00016 Bytes 00124 Bytes 01032 Bytes 011–111 Reserved
TTYPE	<p>Transfer Type.</p> <ul style="list-style-type: none"> 00000 Address-only 00001 Address-only 00010 Single beat or burst write 00011 Reserved 00100 Address-only 00101 Reserved 00110 Burst write 00111 Reserved 01000 Address-only 01001 Address-only 01010 Single beat or burst read 01011 Single beat or burst read 01100 Address-only 01101 Address-only 01110 Burst read 01111 Reserved 10000 Address-only 1XX01 Reserved 10010 Single beat write 1XX11 Reserved 10100 ECOWX—Illegal single beat write 10110 Reserved 11000 Address-only 11010 Single beat or burst read 11100 ECIWX—Illegal single beat read 11110 Burst read

10.2.1.8 Arbiter Event Address Register (AEADR)

Arbiter event address register (AEADR) reports the address of transaction that causes the error, which is specified in the event register. See [Section 10.2.1.4, “Arbiter Event Register \(AER\),”](#) for more information. AEADR is cleared only by power-on reset. The address of the first error event is stored. As AEADR is not effected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the bus is stalled.

[Figure 10-9](#) shows the fields of AEADR.

Offset 0x1cAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. Arbiter Event Address Register (AEADR)

Table 10-9. AEADR Field Descriptions

Field	Description
ADDR	Address of the event, reported in AEATR register. See Section 10.2.1.7, “Arbiter Event Attributes Register (AEATR),” for more information.

10.2.1.9 Arbiter Event Response Register (AERR)

Arbiter event response register (AERR) determines whether different error conditions cause interrupt or reset request. Setting a bit defines the corresponding error condition to cause reset request; clearing a bit defines the corresponding error condition to cause interrupt. [Figure 10-10](#) shows the fields of AERR.

Offset 0x20 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	ETEA	RES	ECW	AO	DTO	ATO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-10. Arbiter Event Response Register (AERR)

Table 10-10. AERR Field Descriptions

Field	Description
ETEA	Transfer error. External \overline{TEA} . Detection of transfer error assertion of \overline{TEA} signal by one of the slaves event response. 0 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves causes interrupt. 1 Detection of transfer error assertion of \overline{TEA} signal by one of the slaves causes reset request.
RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes interrupt. 1 Transaction with reserved transfer type causes reset request.
ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes interrupt. 1 Transaction with external control word transfer type causes reset request.
AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes interrupt. 1 Transaction with address only transfer type causes reset request.
DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes interrupt. 1 Data tenure time out causes reset request.
ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes interrupt. 1 Address tenure time out causes reset request.

10.3 Functional Description

The following sections describe arbiter functionality: arbitration policy and bus error detection.

10.3.1 Arbitration Policy

The arbitration process involves the masters and the arbiter. The masters arbitrate on the privilege to own the address tenure. For the data tenure, the CSB arbiter uses the same order of transactions as address tenures. [Figure 10-11](#) shows the interface signals between the CSB arbiter and masters that are involved in the address bus arbitration.

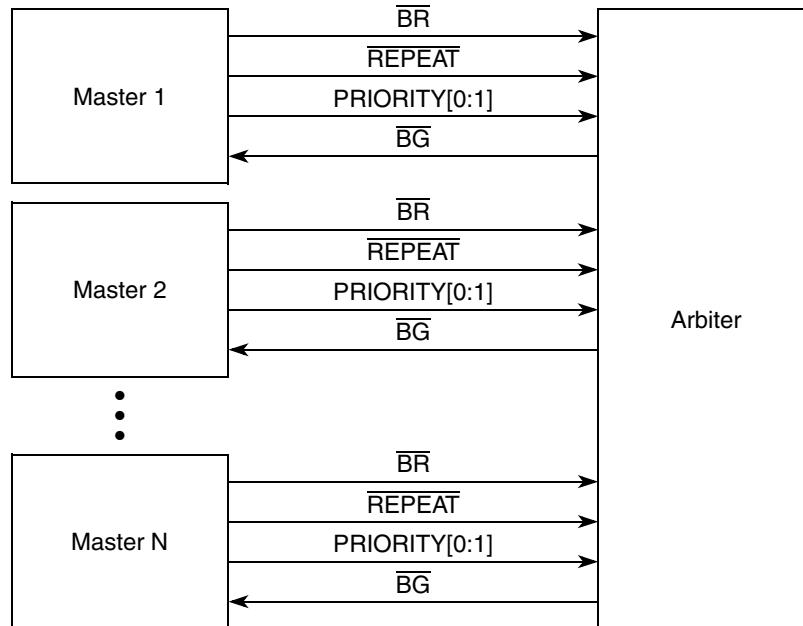


Figure 10-11. Address Bus Arbitration

A master has to acquire the address bus ownership before it starts any transaction. The master asserts its own bus request signal along with the arbitration attribute signals $\overline{\text{REPEAT}}$ & $\text{PRIORITY}[0:1]$. The arbiter later asserts the corresponding address bus grant signal to the requesting master depending on the system states and arbitration scheme. See [Section 10.3.1.1, “Address Bus Arbitration With \$\text{PRIORITY}\[0:1\]\$,”](#) for details information on arbitration scheme. When address bus grant is received the master can start the address tenure.

10.3.1.1 Address Bus Arbitration With PRIORITY[0:1]

When a master asserts its bus request to acquire the address bus ownership, it can drive its PRIORITY[0:1] signals to indicate request priority. The master would be served sooner because of its higher priority level. The arbiter takes this extra information into consideration to yield better service for a higher priority request than a lower priority request. Therefore, the arbiter operates according to the following priority based arbitration scheme:

1. For every priority level fair arbitration scheme is used (a simple Round Robin scheme)
2. For every priority level other than 0, one place is reserved as a place holder for lower level arbitration rings.
3. Each master can change it's priority level at any time.

Figure 10-12 shows an example of priority based arbitration algorithm with 4 priority levels. In this example, if all masters request the bus continuously, the following order of bus grants occurs with the specific bandwidth:

- M6 gets 1/2 of the bus bandwidth
- M4 & M5 each gets 1/6 of the bus bandwidth
- M0 & M3 each gets 1/18 of the bus bandwidth
- M1 & M2 each gets 1/36 of the bus bandwidth

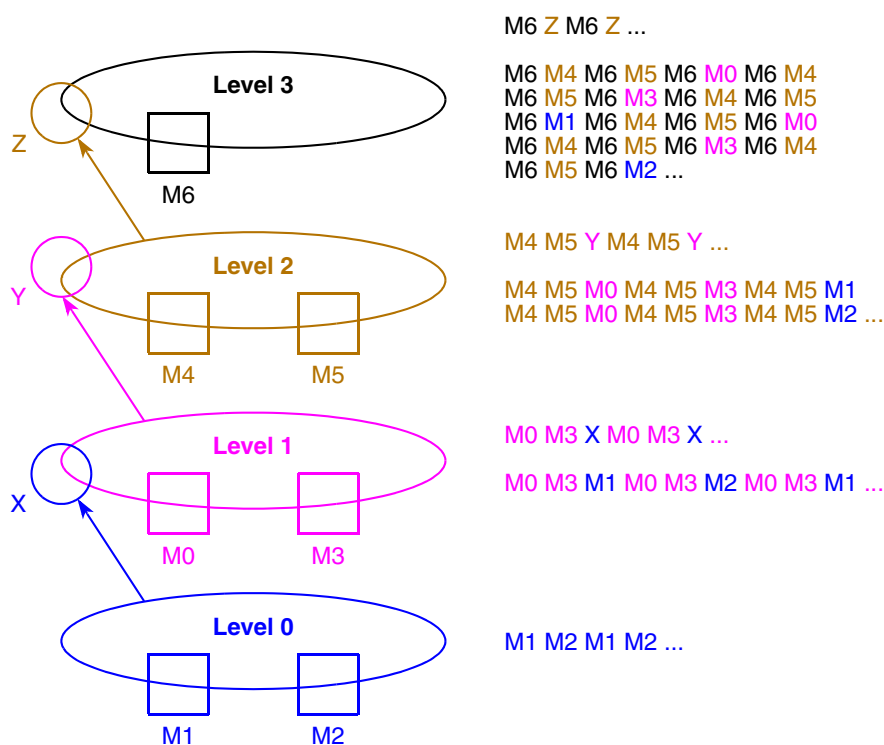


Figure 10-12. An Example of Priority Based Arbitration Algorithm

10.3.1.2 Address Bus Arbitration With $\overline{\text{REPEAT}}$

When a master owns the current address bus and wants to perform another transaction, it can assert bus request along with $\overline{\text{REPEAT}}$, to make a repeat request to the arbiter. Consequently, the arbiter asserts bus grant to the same master if the current address tenure is not being $\overline{\text{ARTRY}}$ 'd. This happens regardless of the priority level of bus request from other masters. In another word, repeat request overrides the priority scheme.

Even though repeat request can improve the page hit ratio and the overall memory bandwidth efficiency, it can increase the worst case latency of individual master. Therefore, the arbiter has programmable counter to limit the maximum number of consecutive transactions that are performed by masters. When the counter expires, arbiter ignores the $\overline{\text{REPEAT}}$ signal and falls back to the regular arbitration scheme. PCI master has a dedicated repeat counter as it might need more repeated transactions before accepting read requests. PCI ordering rules require that the PCI bridge should empty all queued write operations before any new read operation can begin. See Section 3.2.5, Transaction Ordering and Posting, of the *PCI Local Bus Specifications Rev 2.2* for more information.

See [Section 10.2.1.1, “Arbiter Configuration Register \(ACR\),”](#) for more details about programming ACR[RPTCNT] and ACR[PCI_RPTCNT].

10.3.1.3 Address Bus Arbitration After $\overline{\text{ARTRY}}$

The $\overline{\text{ARTRY}}$ protocol is used primarily by the CPU to interrupt a transaction that hits to a modified line in its D-cache, so that it can maintain data coherency by performing the snoop copyback. In addition, any master and/or slave can $\overline{\text{ARTRY}}$ a transaction for whatever reason.

When an address tenure is $\overline{\text{ARTRY}}$ 'd, all masters must negate their bus request signals during the cycle after $\overline{\text{ARTRY}}$ except the master, that asserted $\overline{\text{ARTRY}}$ signal. In addition, PCI master is allowed to request the bus, if the PCI slave asserted $\overline{\text{ARTRY}}$ signal. The cycle after $\overline{\text{ARTRY}}$ is called WOP (Window Of Opportunity).

During the WOP cycle, the arbiter performs the arbitration the same way as in regular arbitration cycle, except parking master policy does not apply. When CPU asserts $\overline{\text{ARTRY}}$, the bus is immediately granted to the CPU to perform snoop copyback. After the completion of snoop copyback, the arbiter grants the bus back to the master that had its transaction $\overline{\text{ARTRY}}$ 'd. If $\overline{\text{ARTRY}}$ was asserted by any other master or slave, the arbiter performs the arbitration the same way as it would perform it for regular arbitration cycle. The master, that had its transaction $\overline{\text{ARTRY}}$ 'ed by any master or slave except CPU, is put at the end of priority list. It can be programmed to park the bus to CPU on WOP cycle or not to park it to any master. Refer to [Section 10.2.1.1, “Arbiter Configuration Register \(ACR\),”](#) for more detail about ACR[WPARK].

10.3.1.4 Address Bus Parking

The arbiter supports address bus parking. This feature implies that when no master is requesting the bus (all bus requests are negated), the arbiter can choose to park the address bus (or assert the address bus grant) to a master. The parked master can skip the bus request and assume the bus mastership directly. This reduces the access latency for parked master.

See [Section 10.2.1.1, “Arbiter Configuration Register \(ACR\),”](#) for more details about ACR[APARK] and ACR[PARKM].

10.3.1.5 Data Bus Arbitration

For every committed address tenure, if the transfer type indicates that it is not address-only or reserved-type transaction, a data tenure is required to complete the transaction.

In the MPC5121e system, the arbiter controls the issuing of data bus grants to a master and a slave, which are involved in a data tenure of a previously performed address tenure.

- In data tenure without data streaming (master or slave or both doesn't support data streaming), the arbiter guarantees that the data bus grants is asserted only when the data bus is idle.
- In data tenure with data streaming (both master and slave support data streaming), the arbiter guarantees that the data bus grants is asserted when the data bus is either idle or waiting for last transfer acknowledge.

10.3.2 Bus Error Detection

The arbiter is responsible for tracking the following cases on the bus:

- Address time out
- Data time out
- Transfer error External TEA
- Address only transaction type
- Reserved transaction type
- Illegal (ECWIX/ECWOX) transaction type

10.3.2.1 Address Time Out

Address time out occurs, if the address tenure was not ended before the specified time-out period (programmed by ATR[ATO]) expires between the assertion of the \overline{TS} signal until the assertion of the \overline{ACK} signal. In this case, the arbiter performs as follows:

1. Ends the address tenure by asserting \overline{AACK} .
2. Starts data tenure and ends it by asserting transfer error \overline{TEA} .
3. Reports on the event to AER[ATO] if reporting enabled by ATER[ATO].
4. Issues reset request, MCP or regular interrupt according to AERR[ATO] and AIDR[ATO] if enabled by AMR[ATO] and if reporting enabled by ATER[ATO].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

10.3.2.2 Data Time Out

Data time out occurs, if the data tenure was not ended before the specified time-out period (programmed by ATR[DTO]) expires between the assertion of $\overline{\text{DBB}}$ until the assertion of last $\overline{\text{TA}}$. In this case, the arbiter performs as follows:

1. Ends the data tenure by asserting transfer error $\overline{\text{TEA}}$.
2. Reports on this event in AER[DTO] if reporting enabled by ATER[DTO].
3. Issues reset request, MCP or regular interrupt according to AERR[DTO] and AIDR[DTO], if enabled by AMR[DTO] and if reporting enabled by ATER[DTO].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

10.3.2.3 Transfer errorExternal TEA

The arbiter tracks the transfer error $\overline{\text{TEA}}$ signal that is asserted by one of the slaves. In this case, the arbiter performs as follows:

1. Reports on the event to AER[ETEA] if reporting enabled by ATER[ETEA].
2. Issues reset request, MCP or regular interrupt according to AERR[ETEA] and AIDR[ETEA] if enabled by AMR[ETEA] and if reporting enabled by ATER[ETEA].
3. Updates transaction attributes and address of AEATR and AEADR for the first error event.

10.3.2.4 Address Only Transaction Type

Table 10-11 shows transaction types, which are defined as address only:

Table 10-11. Address Only Transaction Type Encoding

TTYPE[0:4]	Bus Command
00000	Clean block
00100	Flush block
01000	Sync
01100	Kill block
10000	eieio
11000	TLB Invalidate
00001	lwarx reservation set
01001	tlbsync
01101	icbi

The arbiter allows address-only (AO) transactions on the bus and the e300 core can issue address-only (AO) transactions (see HID0 [ABE] in the Power Architecture Core Family Reference Manual). Because there is no advantage in using AO transaction in this system, the bus monitor allows the detection of AO transactions and treats them as an error.

The transaction with an address only transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
2. Reports on the event to AER[AO] if reporting enabled by ATER[AO].
3. Issues reset request, MCP or regular interrupt according to AERR[AO] and AIDR[AO] if enabled by AMR[AO] and if reporting enabled by ATER[AO].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

10.3.2.5 Reserved Transaction Type

Table 10-12 shows transaction types, which are defined as reserved:

Table 10-12. Reserved Transaction Type Encoding

TTYPE[0:4]	Bus Command
00101	Reserved
1XX01	Reserved for customer
10110	Reserved
00011	Reserved
00111	Reserved
01111	Reserved
1XX11	Reserved for customer

The transaction with a reserved transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
2. Reports on the event to AER[RES] if reporting enabled by ATER[RES].
3. Issues reset request, MCP or regular interrupt according to AERR[RES] and AIDR[RES], if enabled by AMR[RES] and if reporting enabled by ATER[RES].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

10.3.2.6 Illegal (ECIWX/ECOWX) Transaction Type

Table 10-13 shows transaction types, which are defined as illegal.

Table 10-13. Illegal Transaction Type Encoding

TTYPE[0:4]	Bus Command
10100	External control word write (ecowx)
11100	External control word read (eciwx)

The transaction with an illegal (ECIWX, ECOWX) transfer type, the arbiter performs as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
2. Starts data tenure and ends data tenure by asserting $\overline{\text{TEA}}$.
3. Reports on the event in AER[ECW] if reporting enabled by ATER[ECW].
4. Issues reset request, MCP or regular interrupt according to AERR[ECW] and AIDR[ECW], if enabled by AMR[ECW] and if reporting enabled by ATER[ECW].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

See [Section 10.2.1.3, “Arbiter Transfer Error Register \(ATER\),”](#) [Section 10.2.1.4, “Arbiter Event Register \(AER\),”](#) [Section 10.2.1.5, “Arbiter Interrupt Definition Register \(AIDR\),”](#) [Section 10.2.1.6, “Arbiter Mask Register \(AMR\),”](#) [Section 10.2.1.7, “Arbiter Event Attributes Register \(AEATR\),”](#) [Section 10.2.1.8, “Arbiter Event Address Register \(AEADR\),”](#) and [Section 10.2.1.9, “Arbiter Event Response Register \(AERR\)”](#) for more information.

10.4 Initialization/Applications Information

The following sections describe the initialization and error handling sequences for the arbiter:

10.4.1 Initialization Sequence

The following initialization sequence is recommended:

1. Write to ACR register to configure pipeline depth, address bus parking mode, global maximum repeat count, PCI maximum repeat count and address acknowledge wait states.
2. Write to ATER register to define which event is considered error event and which won't.
3. Write to AERR to define whether different error events causes reset request or interrupt.
4. Write to AIDR to define the kind of interrupt (regular or MCP) that is caused by every error event. This is only necessary if interrupts are enabled and AERR defines error events to cause interrupt.
5. Write to AMR to enable interrupts.
6. Write to ATR to set the ATO and DTO timers. This is only necessary if the required timers are less than the maximum value (which is default).

10.4.2 Error Handling Sequence

The following error handling sequence is recommended:

1. Read the AER register to find out about the error that occurred in the system. Also, read the values of AEATR and AEADR to check on the first error event in the system.
2. If those registers are not accessible because of a stalled bus, reset the chip and read the values of the AEATR and AEADR registers to check on the event that causes this problem to the system. Use $\overline{\text{HRESET}}$ to reset the chip to guarantee, that the information stored in AEATR and AEADR is not lost.
3. Clear all the previous events by writing 1's to the AER register. This register is also cleared after reset.

Chapter 11

Direct Memory Access (DMA)

11.1 Introduction

The direct memory access (DMA) module provides a flexible and efficient way to move blocks of data within the system. The DMA controller reduces the workload on the microprocessor, allowing it to continue execution of system software.

The DMA module includes a DMA engine, interfaces to peripheral buses and a dynamic clock gating controller. The DMA engine performs source and destination address calculations, actual data movement operations, and has a local memory containing the transfer control descriptors (TCD) for the channels. The interfaces to peripheral buses grab data from source peripherals or pass data to destination peripherals. The dynamic clock gating controller monitors the DMA engine and interface activities, turns off clocks to the DMA engine, and DMA interfaces after detecting that the bus is idle.

11.1.1 Features

- Programmable multi-channel interface to peripherals
 - Supports 64 channels, 32-bit data path widths
 - Supports data movement of peripherals: MDDRC, PSC_FIFO, SDHC, NFC, PATA, LPC, SPDIF, and MBX (not available in MPC5123).
- Unrestricted data movement within physical memory address space
 - All data movement via dual-address transfers: read from source, write to destination
 - The transfer control descriptors(TCD) supports two-deep, nested transfer operations: An inner data transfer loop defined by a minor byte transfer count, and an outer data transfer loop defined by a major iteration count.
- Support for external DMA request over GPIO interface
 - 31 GPIO pins can be used as DMA requestors
- Flexible protocol programmability
 - Programmable source, destination addresses, transfer size
 - Transfer control descriptors(TCD) organized to support two-deep, nested transfer operations
 - Supports fixed-priority and round-robin channel arbitration
 - Channel completion reported via interrupt requests
 - Supports scatter/gather DMA processing

11.2 Memory Map and Register Definition

Table 11-1. DMA Block Memory Map

Offset	Register				Access	Section/Page
0x0000	DMACR—DMA Control Register				R/W	11.2.1.1/11-6
0x0004	DMAES—DMA Error Status				R	11.2.1.2/11-7
0x0008	DMAERQH—DMA Enable Request High (Channels 63-32)				R/W	11.2.1.3/11-9
0x000c	DMAERQL—DMA Enable Request Low (Channels 31-00)				R/W	11.2.1.3/11-9
0x0010	DMAEEIH—DMA Enable Error Interrupt High (Channels 63-32)				R/W	11.2.1.4/11-11
0x0014	DMAEEIL—DMA Enable Error Interrupt Low (Channels 31-00)				R/W	11.2.1.4/11-11
0x0018	DMASERQ— DMA Set Enable Request	DMACERQ— DMA Clear Enable Request	DMASEEI— DMA Set Enable Error Interrupt	DMACEEI— DMA Clear Enable Error Interrupt	R/W	11.2.1.5/11-13 , 11.2.1.6/11-13 , 11.2.1.7/11-14 , 11.2.1.8/11-14
0x001c	DMACINT— DMA Clear Interrupt Request	DMACERR— DMA Clear Error	DMASSRT— DMA Set Start Bit	DMACDNE— DMA Clear Done Status Bit	R/W	11.2.1.9/11-15 , 11.2.1.10/11-16 , 11.2.1.11/11-16 , 11.2.1.12/11-17
0x0020	DMAINTH—DMA Interrupt Request High (Channels 63-32)				R/W	11.2.1.13/11-17
0x0024	DMAINTL—DMA Interrupt Request Low (Channels 31-00)				R/W	11.2.1.13/11-17
0x0028	DMAERRH—DMA Error High (Channels 63-32)				R/W	11.2.1.14/11-20
0x002c	DMAERRL—DMA Error Low (Channels 31-00)				R/W	11.2.1.14/11-20
0x0030	DMAHRSH—DMA Hardware Request Status High (Channels 63-32)				R	11.2.1.15/11-22
0x0034	DMAHRSL—DMA Hardware Request Status Low (Channels 31-00)				R	11.2.1.15/11-22
0x0038	DMAIHSA—DMA Interrupt High Select AXE (Channels 63-32)				R/W	11.2.1.16/11-23
0x003c	DMAILSA—DMA Interrupt Low Select AXE (Channels 31-00)				R/W	11.2.1.16/11-23
0x0040- 0x00fc	Reserved					
0x0100	DCHPRI0— DMA Channel 0 Priority	DCHPRI1— DMA Channel 1 Priority	DCHPRI2— DMA Channel 2 Priority	DCHPRI3— DMA Channel 3 Priority	R/W	11.2.1.17/11-24
0x0104	DCHPRI4— DMA Channel 4 Priority	DCHPRI5— DMA Channel 5 Priority	DCHPRI6— DMA Channel 6 Priority	DCHPRI7— DMA Channel 7 Priority	R/W	11.2.1.17/11-24
0x0108	DCHPRI8— DMA Channel 8 Priority	DCHPRI9— DMA Channel 9 Priority	DCHPRI10— DMA Channel 10 Priority	DCHPRI11— DMA Channel 11 Priority	R/W	11.2.1.17/11-24
0x010c	DCHPRI12—D MA Channel 12 Priority	DCHPRI13—D MA Channel 13 Priority	DCHPRI14—D MA Channel 14 Priority	DCHPRI15—D MA Channel 15 Priority ()	R/W	11.2.1.17/11-24
0x0110	DCHPRI16—D MA Channel 16 Priority	DCHPRI17—D MA Channel 17 Priority	DCHPRI18—D MA Channel 18 Priority	DCHPRI19—D MA Channel 19 Priority	R/W	11.2.1.17/11-24

Table 11-1. DMA Block Memory Map (continued)

Offset	Register				Access	Section/Page
0x0114	DCHPRI20— DMA Channel 20 Priority	DCHPRI21— DMA Channel 21 Priority	DCHPRI22— DMA Channel 22 Priority	DCHPRI23— DMA Channel 23 Priority	R/W	11.2.1.17/11-24
0x0118	DCHPRI24— DMA Channel 24 Priority	DCHPRI25— DMA Channel 25 Priority	DCHPRI26— DMA Channel 26 Priority	DCHPRI27— DMA Channel 27 Priority	R/W	11.2.1.17/11-24
0x011c	DCHPRI28— DMA Channel 28 Priority	DCHPRI29— DMA Channel 29 Priority	DCHPRI30— DMA Channel 30 Priority	DCHPRI31— DMA Channel 31 Priority	R/W	11.2.1.17/11-24
0x0120	DCHPRI32— DMA Channel 32 Priority	DCHPRI33— DMA Channel 33 Priority	DCHPRI34— DMA Channel 34 Priority	DCHPRI35— DMA Channel 35 Priority	R/W	11.2.1.17/11-24
0x0124	DCHPRI36— DMA Channel 36 Priority	DCHPRI37— DMA Channel 37 Priority	DCHPRI38— DMA Channel 38 Priority	DCHPRI39— DMA Channel 39 Priority	R/W	11.2.1.17/11-24
0x0128	DCHPRI40— DMA Channel 40 Priority	DCHPRI41— DMA Channel 41 Priority	DCHPRI42— DMA Channel 42 Priority	DCHPRI43— DMA Channel 43 Priority	R/W	11.2.1.17/11-24
0x012c	DCHPRI44— DMA Channel 44 Priority	DCHPRI45— DMA Channel 45 Priority	DCHPRI46— DMA Channel 46 Priority	DCHPRI47— DMA Channel 47 Priority	R/W	11.2.1.17/11-24
0x0130	DCHPRI48— DMA Channel 48 Priority	DCHPRI49— DMA Channel 49 Priority	DCHPRI50— DMA Channel 50 Priority	DCHPRI51— DMA Channel 51 Priority	R/W	11.2.1.17/11-24
0x0134	DCHPRI52— DMA Channel 52 Priority	DCHPRI53— DMA Channel 53 Priority	DCHPRI54— DMA Channel 54 Priority	DCHPRI55— DMA Channel 55 Priority	R/W	11.2.1.17/11-24
0x0138	DCHPRI56— DMA Channel 56 Priority	DCHPRI57— DMA Channel 57 Priority	DCHPRI58— DMA Channel 58 Priority	DCHPRI59— DMA Channel 59 Priority	R/W	11.2.1.17/11-24
0x013c	DCHPRI60— DMA Channel 60 Priority	DCHPRI61— DMA Channel 61 Priority	DCHPRI62— DMA Channel 62 Priority	DCHPRI63— DMA Channel 63 Priority	R/W	11.2.1.17/11-24
0x0140-0 x0ffc	Reserved					
0x1000-0 x11fc	TCD00-TCD15				R/W	11.2.1.18/11-25
0x1200-0 x13fc	TCD16-TCD31				R/W	11.2.1.18/11-25
0x1400-0 x15fc	TCD32-TCD47				R/W	11.2.1.18/11-25
0x1600-0 x17fc	TCD48-TCD63				R/W	11.2.1.18/11-25
0x1800-0 x3fff	Reserved					

Table 11-2. DMA Channel Assignments

Requester	DMA Channel	DMA Request Enable	DMA Interrupt Request	DMA Error
GPIO0	DMA_REQ63	ERQ63	INT63	ERR63
GPIO1	DMA_REQ62	ERQ62	INT62	ERR62
GPIO2	DMA_REQ61	ERQ61	INT61	ERR61
GPIO3	DMA_REQ60	ERQ60	INT60	ERR60
GPIO4	DMA_REQ59	ERQ59	INT59	ERR59
GPIO5	DMA_REQ58	ERQ58	INT58	ERR58
GPIO6	DMA_REQ57	ERQ57	INT57	ERR57
GPIO7	DMA_REQ56	ERQ56	INT56	ERR56
GPIO8	DMA_REQ55	ERQ55	INT55	ERR55
GPIO9	DMA_REQ54	ERQ54	INT54	ERR54
GPIO10	DMA_REQ53	ERQ53	INT53	ERR53
GPIO11	DMA_REQ52	ERQ52	INT52	ERR52
GPIO12	DMA_REQ51	ERQ51	INT51	ERR51
GPIO13	DMA_REQ50	ERQ50	INT50	ERR50
GPIO14	DMA_REQ49	ERQ49	INT49	ERR49
GPIO15	DMA_REQ48	ERQ48	INT48	ERR48
GPIO16	DMA_REQ47	ERQ47	INT47	ERR47
GPIO17	DMA_REQ46	ERQ46	INT46	ERR46
GPIO18	DMA_REQ45	ERQ45	INT45	ERR45
GPIO19	DMA_REQ44	ERQ44	INT44	ERR44
GPIO20	DMA_REQ43	ERQ43	INT43	ERR43
GPIO21	DMA_REQ42	ERQ42	INT42	ERR42
GPIO22	DMA_REQ41	ERQ41	INT41	ERR41
GPIO23	DMA_REQ40	ERQ40	INT40	ERR40
GPIO24	DMA_REQ39	ERQ39	INT39	ERR39
GPIO25	DMA_REQ38	ERQ38	INT38	ERR38
GPIO26	DMA_REQ37	ERQ37	INT37	ERR37
GPIO27	DMA_REQ36	ERQ36	INT36	ERR36
GPIO28	DMA_REQ35	ERQ35	INT35	ERR35
GPIO29	DMA_REQ34	ERQ34	INT34	ERR34
GPIO30	DMA_REQ33	ERQ33	INT33	ERR33
MDDRC	DMA_REQ32	ERQ32	INT32	ERR32
MBX ¹	DMA_REQ31	ERQ31	INT31	ERR31

Table 11-2. DMA Channel Assignments (continued)

Requester	DMA Channel	DMA Request Enable	DMA Interrupt Request	DMA Error
SDHC	DMA_REQ30	ERQ30	INT30	ERR30
NFC	DMA_REQ29	ERQ29	INT29	ERR29
PATA_TX_FIFO_ALARM	DMA_REQ28	ERQ28	INT28	ERR28
PATA_RX_FIFO_ALARM	DMA_REQ27	ERQ27	INT27	ERR27
LPC	DMA_REQ26	ERQ26	INT26	ERR26
SPDIF_RX	DMA_REQ25	ERQ25	INT25	ERR25
SPDIF_TX	DMA_REQ24	ERQ24	INT24	ERR24
PSC_FIFO_TX11	DMA_REQ23	ERQ23	INT23	ERR23
PSC_FIFO_TX10	DMA_REQ22	ERQ22	INT22	ERR22
PSC_FIFO_TX9	DMA_REQ21	ERQ21	INT21	ERR21
PSC_FIFO_TX8	DMA_REQ20	ERQ20	INT20	ERR20
PSC_FIFO_TX7	DMA_REQ19	ERQ19	INT19	ERR19
PSC_FIFO_TX6	DMA_REQ18	ERQ18	INT18	ERR18
PSC_FIFO_TX5	DMA_REQ17	ERQ17	INT17	ERR17
PSC_FIFO_TX4	DMA_REQ16	ERQ16	INT16	ERR16
PSC_FIFO_TX3	DMA_REQ15	ERQ15	INT15	ERR15
PSC_FIFO_TX2	DMA_REQ14	ERQ14	INT14	ERR14
PSC_FIFO_TX1	DMA_REQ13	ERQ13	INT13	ERR13
PSC_FIFO_TX0	DMA_REQ12	ERQ12	INT12	ERR12
PSC_FIFO_RX11	DMA_REQ11	ERQ11	INT11	ERR11
PSC_FIFO_RX10	DMA_REQ10	ERQ10	INT10	ERR10
PSC_FIFO_RX9	DMA_REQ9	ERQ9	INT9	ERR9
PSC_FIFO_RX8	DMA_REQ8	ERQ8	INT8	ERR8
PSC_FIFO_RX7	DMA_REQ7	ERQ7	INT7	ERR7
PSC_FIFO_RX6	DMA_REQ6	ERQ6	INT6	ERR6
PSC_FIFO_RX5	DMA_REQ5	ERQ5	INT5	ERR5
PSC_FIFO_RX4	DMA_REQ4	ERQ4	INT4	ERR4
PSC_FIFO_RX3	DMA_REQ3	ERQ3	INT3	ERR3
PSC_FIFO_RX2	DMA_REQ2	ERQ2	INT2	ERR2
PSC_FIFO_RX1	DMA_REQ1	ERQ1	INT1	ERR1
PSC_FIFO_RX0	DMA_REQ0	ERQ0	INT0	ERR0

¹ Not available in MPC5123. In this don't enable DMA_REQ31 and corresponding interrupt requests.

11.2.1 Register Descriptions

11.2.1.1 DMA Control Register (DMACR)

The 32-bit DMA control register (DMACR) defines the basic operating configuration of the DMA.

The DMA arbitrates channel service requests in groups of 16 channels. The 64 channel configurations have four groups (3, 2, 1, and 0). Group 3 contains channels 63-48. Group 2 contains channels 47-32. Group 1 contains channels 31-16. Group 0 contains channels 15-0.

Arbitration within a group can be configured to use a fixed priority or a round robin. In fixed priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. In round robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through without regard to priority.

The group priorities operate in a similar fashion. In group fixed priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 3 is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRPnPri registers. All group priorities must have unique values before any channel service requests occur. Otherwise, a configuration error is reported. Unused group priority registers, per configuration, are unimplemented in the DMACR. In group round robin mode, the group priorities are ignored and the groups are cycled through without regard to priority.

Offset: 0x0000

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EDCG	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	GRP3PRI	GRP2PRI	GRP1PRI	GRP0PRI	0	0	0	0	0	0	0	0	ERGA	ERCA	EDBG	0
W																
Reset	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 11-1. DMA Control Register (DMACR) (continued)

Table 11-3. DMACR Field Descriptions

Field	Description
EDCG	Enable Clock Dynamic Gating 0 Disable Clock Dynamic Gating 1 Enable Clock Dynamic Gating
GRP3PRI	Channel Group 3 Priority. Group 3 priority level when fixed priority group arbitration is enabled.
GRP2PRI	Channel Group 2 Priority. Group 2 priority level when fixed priority group arbitration is enabled.
GRP1PRI	Channel Group 1 Priority. Group 1 priority level when fixed priority group arbitration is enabled.

Table 11-3. DMACR Field Descriptions

Field	Description
GRP0PRI	Channel Group 0 Priority. Group 0 priority level when fixed priority group arbitration is enabled.
ERGA	Enable Round Robin Group Arbitration 0 Fixed priority arbitration is used for selection among the groups. 1 Round robin arbitration is used for selection among the groups.
ERCA	Enable Round Robin Channel Arbitration 0 Fixed priority arbitration is used for channel selection within each group. 1 Round robin arbitration is used for channel selection within each group.
EDBG	Enable Debug 0 Ignore DMA debug input. 1 Setting of TEST dma_dbg bit, AXE halt, or e300 breakpoint causes the DMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when EST dma_dbg bit is cleared, AXE resumes execution, e300 resumes execution, or the EDBG bit is cleared. Note: AXE and e300 halt need to be enabled by separate TEST dma_halt_en bits.

11.2.1.2 DMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit does not equal the TCD.BITER.E_LINK bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and asserts an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the DMA_ENGINE with the current source address, destination address, and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction already pipelined after errant access, has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

Direct Memory Access (DMA)

The occurrence of any type of error causes the DMA_ENGINE to immediately stop, and the appropriate channel bit in the DMA error register to be asserted. At the same time, the details of the error condition are loaded into the DMAES register. The major loop complete indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected.

Offset: 0x0004

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	GPE	CPE	ERRCHN[5:0]						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 11-2. DMA Error Status Register (DMAES)

Table 11-4. DMAES Field Descriptions

Field	Description
VLD	Logical OR of all DMAERRH and DMAERRL status bits. 0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
GPE	Group Priority Error 0 No group priority error. 1 The last recorded error was a configuration error among the group priorities. All group priorities are not unique.
CPE	Channel Priority Error 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique.
ERRCHN[5:0]	Error Channel Number The channel number of the last recorded error (excluding GPE and CPE errors).
SAE	Source Address Error 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field. TCD.SADDR is inconsistent with TCD.SSIZE.

Table 11-4. DMAES Field Descriptions (continued)

Field	Description
SOE	Source Offset Configuration 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field. TCD.SOFF is inconsistent with TCD.SSIZE.
DAE	Destination Address Error 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field. TCD.DADDR is inconsistent with TCD.DSIZE.
DOE	Destination Offset Error 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field. TCD.DOFF is inconsistent with TCD.DSIZE.
NCE	Nbytes/Citer Configuration Error 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields. TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or TCD.CITER is equal to zero, or TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
SGE	Scatter/Gather Configuration Error 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.E_SG is enabled. TCD.DLAST_SGA is not on a 32-byte boundary.
SBE	Source Bus Error 0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

11.2.1.3 DMA Enable Request (DMAERQH, DMAERQL)

The DMAERQ{H,L} registers provide a bit map for the implemented 64 channels to enable the request signal for each channel. DMAERQH supports channels 63-32, while DMAERQL covers channels 31-00. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the DMASERQ and DMACERQ registers. The DMA{S,C}ERQ registers are provided so that the request enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the DMAERQ{H,L} registers.

The DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the DMA enable request flag does not affect a channel service request made explicitly through software or a linked channel request.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the DMAERQ bit for that channel. If the

Direct Memory Access (DMA)

TCD.D_REQ bit is set, the corresponding DMAERQ bit is cleared, disabling the DMA request. If the D_REQ bit is cleared, the state of the DMAERQ bit is unaffected.

Offset: 0x0008

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ERQ63	ERQ62	ERQ61	ERQ60	ERQ59	ERQ58	ERQ57	ERQ56	ERQ55	ERQ54	ERQ53	ERQ52	ERQ51	ERQ50	ERQ49	ERQ48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERQ47	ERQ46	ERQ45	ERQ44	ERQ43	ERQ42	ERQ41	ERQ40	ERQ39	ERQ38	ERQ37	ERQ36	ERQ35	ERQ34	ERQ33	ERQ32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x000C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ERQ31	ERQ30	ERQ29	ERQ28	ERQ27	ERQ26	ERQ25	ERQ24	ERQ23	ERQ22	ERQ21	ERQ20	ERQ19	ERQ18	ERQ17	ERQ16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-3. DMA Enable Request Registers (DMAERQH and DMAERQL)

Table 11-5. DMAERQH and DMAERQL Field Descriptions

Field	Description
ERQn	Enable DMA Request n 0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

11.2.1.4 DMA Enable Error Interrupt (DMAEEIH, DMAEEIL)

The DMAEEI{H,L} registers provide a bit map for the implemented 64 channels to enable the error interrupt signal for each channel. DMAEEIH supports channels 63-32, while DMAEEIL covers channels 31-00. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to the DMASEEI and DMACEEI registers. The DMA{S,C}EEI registers are provided so that the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the DMAEEI{H,L} registers.

The DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See [Figure 11-4](#) and [Table 11-6](#) for the DMAEEI definition.

Direct Memory Access (DMA)

Offset: 0x0010

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EEI63	EEI62	EEI61	EEI60	EEI59	EEI58	EEI57	EEI56	EEI55	EEI54	EEI53	EEI52	EEI51	EEI50	EEI49	EEI48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EEI47	EEI46	EEI45	EEI44	EEI43	EEI42	EEI41	EEI40	EEI39	EEI38	EEI37	EEI36	EEI35	EEI34	EEI33	EEI32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x0014

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-4. DMA Enable Error Registers (DMAEEIH and DMAEEIL)

Table 11-6. DMAEEIH and DMAEEIL Field Descriptions

Field	Description
EEIn	Enable Error Interrupt n 0 The error signal for channel n does not generate an error interrupt. 1 The assertion of the error signal for channel n generate an error interrupt request.

11.2.1.5 DMA Set Enable Request (DMASERQ)

The DMASERQ register provides a simple memory-mapped mechanism to set a given bit in the DMAERQ{H,L} registers to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ{H,L} register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAERQ{H,L} to be asserted. Reads of this register return all zeroes. See [Figure 11-5](#) and [Table 11-7](#) for the DMASERQ definition.

Offset: 0x0018

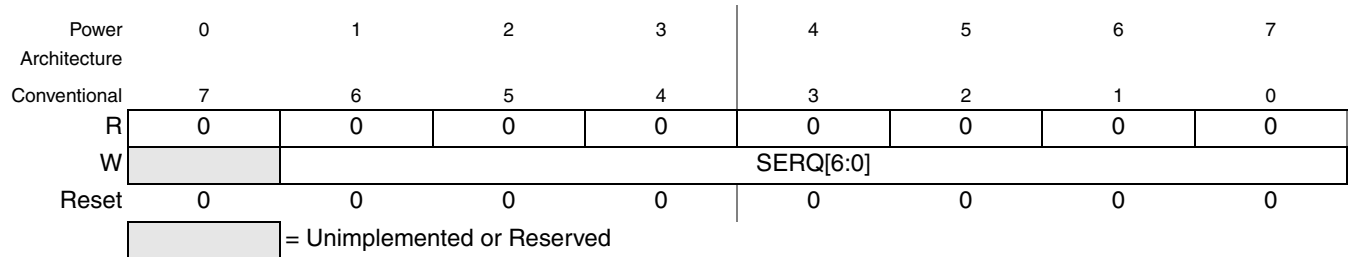


Figure 11-5. DMA Set Enable Request Register (DMASERQ)

Table 11-7. DMASERQ Field Descriptions

Field	Description
SERQ[6:0]	Set Enable Request 0-63 Set the corresponding bit in DMAERQ{H,L} 64-127 Set all bits in DMAERQ{H,L}

11.2.1.6 DMA Clear Enable Request (DMACERQ)

The DMACERQ register provides a simple memory-mapped mechanism to clear a given bit in the DMAERQ{H,L} registers to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERQ{H,L} to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See [Figure 11-6](#) and [Table 11-8](#) for the DMACERQ definition.

Direct Memory Access (DMA)

Offset: 0x0019

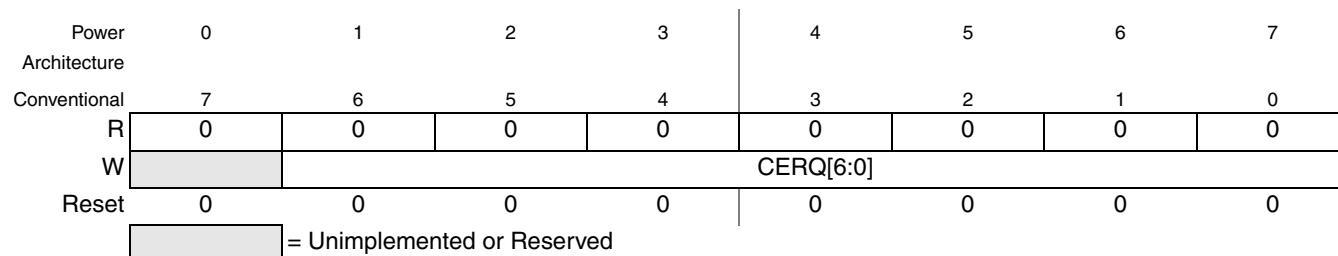


Figure 11-6. DMA Clear Enable Request Register (DMACERQ)

Table 11-8. DMACERQ Field Descriptions

Field	Description
CERQ[6:0]	Clear Enable Request 0-63 Clear corresponding bit in DMAERQ{H,L} 64-127 Clear all bits in DMAERQ{H,L}

11.2.1.7 DMA Set Enable Error Interrupt (DMASEEI)

The DMASEEI register provides a simple memory-mapped mechanism to set a given bit in the DMAEEI{H,L} registers to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI{H,L} register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEI{H,L} to be asserted. Reads of this register return all zeroes. See [Figure 11-7](#) and [Table 11-9](#) for the DMASEEI definition.

Offset: 0x001a

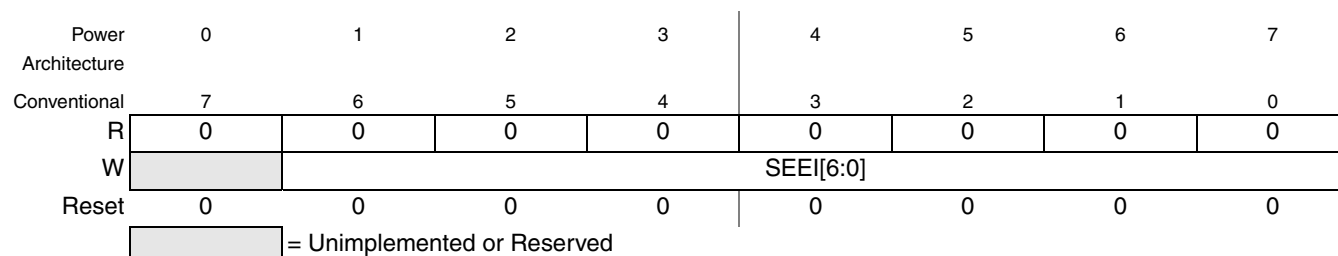


Figure 11-7. DMA Set Enable Error Interrupt Register (DMASEEI)

Table 11-9. DMASEEI Field Descriptions

Field	Description
SEEI[6:0]	Set Enable Error Interrupt 0-63 Set the corresponding bit in DMAEEI{H,L} 64-127 Set all bits in DMAEEI{H,L}

11.2.1.8 DMA Clear Enable Error Interrupt (DMACEEI)

The DMACEEI register provides a simple memory-mapped mechanism to clear a given bit in the DMAEEI{H,L} registers to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire

contents of the DMAEEI{H,L} to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See [Figure 11-8](#) and [Table 11-10](#) for the DMACEEI definition.

Offset: 0x0018

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SERQ[6:0]						
Reset	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 11-8. DMA Clear Enable Error Interrupt Register (DMACEEI)

Table 11-10. DMASEEI Field Descriptions

Field	Description
SERQ[6:0]	Clear Enable Error Interrupt 0-63 Clear corresponding bit in DMAEEI{H,L} 64-127 Clear all bits in DMAEEI{H,L}

11.2.1.9 DMA Clear Interrupt Request (DMACINT)

The DMACINT register provides a simple memory-mapped mechanism to clear a given bit in the DMAINT{H,L} registers to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMAINT{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAINT{H,L} to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes. See [Figure 11-9](#) and [Table 11-11](#) for the DMACINT definition.

Offset: 0x001c

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CINT[6:0]						
Reset	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 11-9. DMA Clear Interrupt Request Register (DMACINT)

Table 11-11. DMACINT Field Descriptions

Field	Description
CINT[6:0]	Clear Interrupt Request 0-63 Clear corresponding bit in DMAINT{H,L} 64-127 Clear all bits in DMAINT{H,L}

11.2.1.10 DMA Clear Error (DMACERR)

The DMACEER register provides a simple memory-mapped mechanism to clear a given bit in the DMAERR{H,L} registers to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMAERR{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERR{H,L} to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See [Figure 11-10](#) and [Table 11-12](#) for the DMACERR definition.

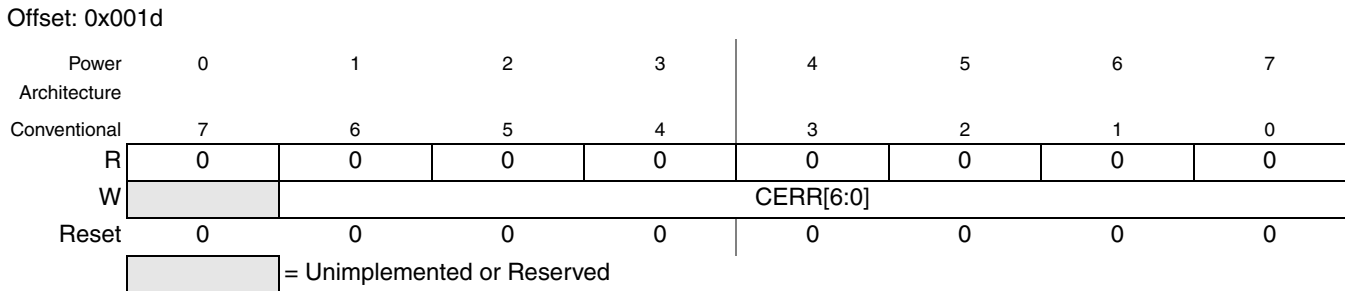


Figure 11-10. DMA Clear Error Register (DMACERR)

Table 11-12. DMACERR Field Descriptions

Field	Description
CERR[6:0]	Clear Error Indicator 0-63 Clear corresponding bit in DMAERR{H,L} 64-127 Clear all bits in DMAERR{H,L}

11.2.1.11 DMA Set START Bit (DMASSRT)

The DMASSRT register provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes. See [Figure 11-11](#) for the TCD START bit definition.

Offset: 0x001e

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SSRT[6:0]						
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 11-11. DMA Set START Bit Register (DMASSRT)

Table 11-13. DMASSRT Field Descriptions

Field	Description
SSRT[6:0]	Set START Bit (Channel Service Request) 0-63 Set the corresponding channel's TCD.start 64-127 Set all TCD.start bits

11.2.1.12 DMA Clear DONE Status (DMACDNE)

The DMACDNE register provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding Transfer Control Descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. Reads of this register return all zeroes. See [Figure 11-12](#) for the TCD DONE bit definition.

Offset: 0x001f

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CDNE[6:0]						
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 11-12. DMA Clear DONE Status Register (DMACDNE)

Table 11-14. DMACDNE Field Descriptions

Field	Description
CDNE[6:0]	Clear DONE Status Bit 0-63 Clear corresponding channel's DONE bit 64-127 Clear all TCD DONE bits

11.2.1.13 DMA Interrupt Request (DMAINTH, DMAINTL)

The DMAINT{H,L} registers provide a bit map for the implemented 64 channels signaling the presence of an interrupt request for each channel. DMAINTH supports channels 63-32, while DMAINTL covers channels 31-00. The DMA_ENGINE signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in

this register. The outputs of this register are directly routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is the software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the DMAINT register. On writes to the DMAINT, a one in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no effect on the corresponding channel's current interrupt status. The DMACINT register is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to the DMAINT{H,L} registers.

See [Figure 11-13](#) and [Table 11-15](#) for the DMAINT definition.

Offset: 0x0020

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INT63	INT62	INT61	INT60	INT59	INT58	INT57	INT56	INT55	INT54	INT53	INT52	INT51	INT50	INT49	INT48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT47	INT46	INT45	INT44	INT43	INT42	INT41	INT40	INT39	INT38	INT37	INT36	INT35	INT34	INT33	INT32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x0024

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT15	INT14	INT13	INT12	INT11	INT10	INT09	INT08	INT07	INT06	INT05	INT04	INT03	INT02	INT01	INT00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-13. DMA Interrupt Request Registers (DMAINTH and DMAINTL)

Table 11-15. DMAINTH and DMAINTL Field Descriptions

Field	Description
INTn	DMA Interrupt Request n 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

11.2.1.14 DMA Error (DMAERRH, DMAERRL)

The DMAERR{H,L} registers provide a bit map for the implemented 64 channels signaling the presence of an error for each channel. DMAERRH supports channels 63-32, while DMAERRL covers channels 31-00. The DMA_ENGINE signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, then logically summed across groups of 16, 32 and 64 channels to form several group error interrupt requests routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the DMACERR register in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the DMAEEI register. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the DMACERR register. On writes to the DMAERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The DMACERR register is provided so the error indicator for a single channel can easily be cleared. See [Figure 11-14](#) and [Table 11-16](#) for the DMAERR definition.

Offset: 0x0028

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ERR63	ERR62	ERR61	ERR60	ERR59	ERR58	ERR57	ERR56	ERR55	ERR54	ERR53	ERR52	ERR51	ERR50	ERR49	ERR48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR47	ERR46	ERR45	ERR44	ERR43	ERR42	ERR41	ERR40	ERR39	ERR38	ERR37	ERR36	ERR35	ERR34	ERR33	ERR32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x002c

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ERR31	ERR30	ERR29	ERR28	ERR27	ERR26	ERR25	ERR24	ERR23	ERR22	ERR21	ERR20	ERR19	ERR18	ERR17	ERR16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-14. DMA Error Registers (DMAERRH and DMAERRL)

Table 11-16. DMAERRH and DMAERRL Field Descriptions

Field	Description
ERRn	DMA Error n 0 An error in channel n has not occurred. 1 An error in channel n has occurred.

11.2.1.15 DMA Hardware Request Status (DMAHRSH, DMAHRSL)

The DMAHRS{H,L} registers provide a bit map for the implemented 64 channels' current hardware request status. DMAHRSH supports channels 63-32, while DMAHRSL covers channels 31-00. Hardware request status reflects the current state of the registered and qualified (via DMAERQ field) request lines as seen by DMA arbitration logic. This view into the hardware request signals may be used for debug purposes.

See [Figure 11-15](#) and [Table 11-17](#) for the DMAHRS definition.

Offset: 0x0030

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HRS63	HRS62	HRS61	HRS60	HRS59	HRS58	HRS57	HRS56	HRS55	HRS54	HRS53	HRS52	HRS51	HRS50	HRS49	HRS48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HRS47	HRS46	HRS45	HRS44	HRS43	HRS42	HRS41	HRS40	HRS39	HRS38	HRS37	HRS36	HRS35	HRS34	HRS33	HRS32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x0034

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HRS31	HRS30	HRS29	HRS28	HRS27	HRS26	HRS25	HRS24	HRS23	HRS22	HRS21	HRS20	HRS19	HRS18	HRS17	HRS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HRS15	HRS14	HRS13	HRS12	HRS11	HRS10	HRS09	HRS08	HRS07	HRS06	HRS05	HRS04	HRS03	HRS02	HRS01	HRS00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-15. DMA Hardware Request Status Registers (DMAHRSH and DMHRS�)

Table 11-17. DMAHRSH and DMAHRSL Field Descriptions

Field	Description
HSA _n	DMA Hardware Request Status 0 A Hardware service request for channel n is not present. 1 A Hardware service request for channel n is present. Note: The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the DMAERQ _n bit.

11.2.1.16 DMA Interrupt Select AXE (DMAIHSA, DMAILSA)

The DMAI{H,L}SA registers provide a bit map for the implemented 64 channels' interrupt direction. When a bit in these two registers is set, the corresponding channel's interrupt is directed to AXE. Otherwise, the interrupt is directed to IPIC. DMAIHSA supports channels 63-32, while DMAILSA covers channels 31-00.

See [Figure 11-16](#) and [Table 11-18](#) for the DMAHRS definition.

Direct Memory Access (DMA)

Offset: 0x0038

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ISA63	ISA62	ISA61	ISA60	ISA59	ISA58	ISA57	ISA56	ISA55	ISA54	ISA53	ISA52	ISA51	ISA50	ISA49	ISA48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ISA47	ISA46	ISA45	ISA44	ISA43	ISA42	ISA41	ISA40	ISA39	ISA38	ISA37	ISA36	ISA35	ISA34	ISA33	ISA32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset: 0x003c

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ISA31	ISA30	ISA29	ISA28	ISA27	ISA26	ISA25	ISA24	ISA23	ISA22	ISA21	ISA20	ISA19	ISA18	ISA17	ISA16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ISA15	ISA14	ISA13	ISA12	ISA11	ISA10	ISA09	ISA08	ISA07	ISA06	ISA05	ISA04	ISA03	ISA02	ISA01	ISA00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-16. DMA Interrupt Select AXE Registers (DMAIHS and DMAIHL)

Table 11-18. DMAIHS and DMAIHL Field Descriptions

Field	Description
ISAn	DMA Interrupt n selects AXE 0 The interrupt of channel n is directed to IPIC 1 The interrupt of channel n is directed to AXE.

11.2.1.17 DMA Channel n Priority (DCHPRIn), n = 0,..., {15,31,63}

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value (0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc.). Software must program the channel priorities with unique values; otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the DCHPRIn register reflect the current priority level of the group of channels in which the

corresponding channel resides. GRPPRI bits are not affected by writes to the DCHPRIn registers. The group priority is assigned in the DMACR. See [Figure 11-1](#) and [Table 11-3](#) for the DMACR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRIn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for group and channel arbitration modes. See [Figure 11-17](#) and [Table 11-19](#) for the DCHPRIn definition.

Offset: 0x100 + n

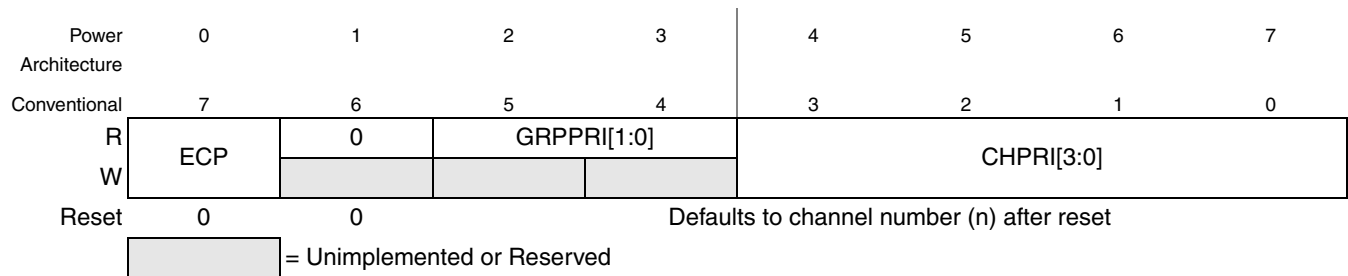


Figure 11-17. DMA Channel n Priority Register (DCHPRIn)

Table 11-19. DCHPRIn Field Descriptions

Field	Description
ECP	Enable Channel Preemption 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
GRPPRI[1:0]	Channel n Current Group Priority. Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read only; writes are ignored.
CHPRI[3:0]	Channel n Arbitration Priority. Channel priority when fixed-priority arbitration is enabled.

11.2.1.18 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order. The definitions of the TCD are presented as eight 32-bit values. [Table 11-20](#) is a 32-bit view of the basic TCD structure.

Table 11-20. TCDn 32-bit Memory Structure

DMA Offset	TCDn Field	
0x1000 + (32 x n) + 0x00	Source Address (saddr)	
0x1000 + (32 x n) + 0x04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
0x1000 + (32 x n) + 0x08	Inner Minor Byte Count (nbytes)	
0x1000 + (32 x n) + 0x0c	Last Source Address Adjustment (slast)	

Table 11-20. TCDn 32-bit Memory Structure

$0x1000 + (32 \times n) + 0x10$	Destination Address (daddr)	
$0x1000 + (32 \times n) + 0x14$	Current Major Iteration Count (citer)	Signed Destination Address Offset (doff)
$0x1000 + (32 \times n) + 0x18$	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
$0x1000 + (32 \times n) + 0x1c$	Beginning Major Iteration Count (biter)	Channel Control/Status (BWC, MAJOR.LINKCH, DONE, ACTIVE, MAJOR.E_LINK, E_SG, D_REQ, INT_HALF, INT_MAJ, START)

Figure 11-18 and Table 11-21 define word 0 of the TCDn structure, the saddr field.

DMA_Offset + $0x1000 + (32 \times n) + 0x00$

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SADDR[31:16]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SADDR[15:00]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 11-18. TCDn Word 0 (TCDn.saddr) Field**Table 11-21. TCDn Word 0 (TCDn.saddr) Field Descriptions**

Field	Description
SADDR[31:0]	Source Address. Memory address pointing to the source data.

Figure 11-19 and Table 11-22 define word one of the TCDn structure, the SOFF and transfer attribute fields.

$$\text{DMA_Offset} + 0x1000 + (32 \times n) + 0x04$$

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SMOD[4:0]				SSIZE[2:0]				DMOD[4:0]				DSIZE[2:0]			
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SOFF[15:0]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 11-19. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) Field

Table 11-22. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) Field Descriptions

Field	Description
SMOD[4:0]	Source address modulo 0 Source address modulo feature is disabled. non-0 The value defines a specific address bit selected to be the value after SADDR + SOFF calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-two size bytes, the queue should be based at a zero-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as $((1 \ll \text{smod}[4:0]) - 1)$ where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a zero-modulo-size range.
SSIZE[2:0]	Source data transfer size 000 8-bit 001 16-bit 010 32-bit 011 Reserved 100 16-byte 101 32-byte 110 Reserved 111 Reserved The attempted specification of a reserved source size produces a configuration error.
DMOD[4:0]	Destination address modulo. See the SMOD[5:0] definition.
DSIZE[2:0]	Destination data transfer size. See the SSIZE[2:0] definition.
SOFF[15:0]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

Figure 11-20 and Table 11-23 define word two of the TCDn structure, the NBYTES field.

DMA_Offset + 0x1000 + (32 x n) + 0x08

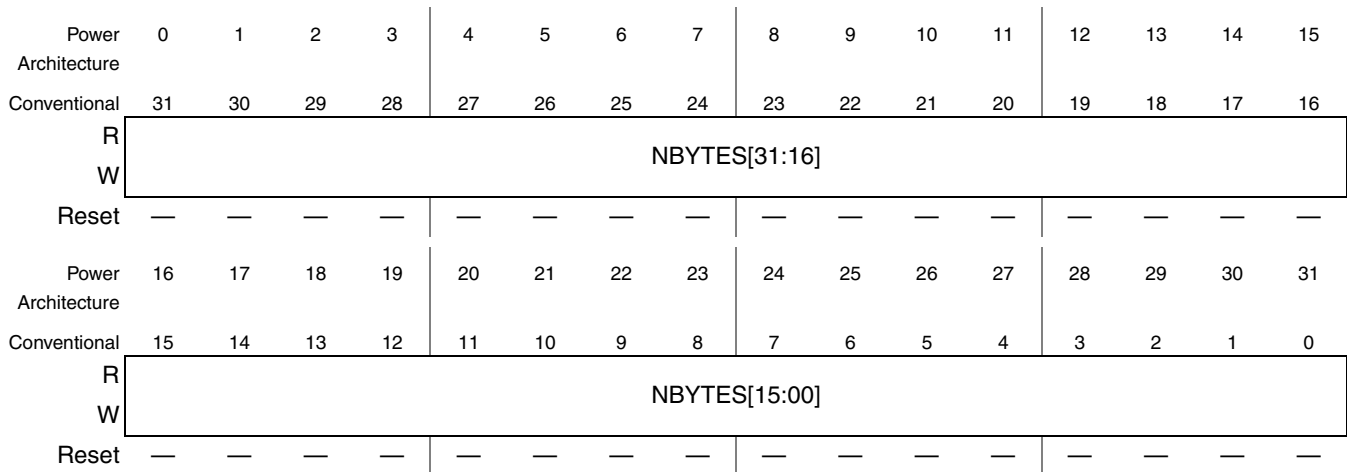


Figure 11-20. TCDn Word 2 (TCDn.nbytes) Field

Table 11-23. TCDn Word 2 (TCDn.nbytes) Field Descriptions

Field	Description
NBYTES[31:0]	<p>Inner Minor Byte Transfer Count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA_ENGINE, and the appropriate reads and writes perform until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.</p> <p>The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4-Gbyte transfer.</p>

Figure 11-21 and Table 11-24 define word three of the TCDn structure, the SLAST field.

DMA_Offset + 0x1000 + (32 x n) + 0x0c

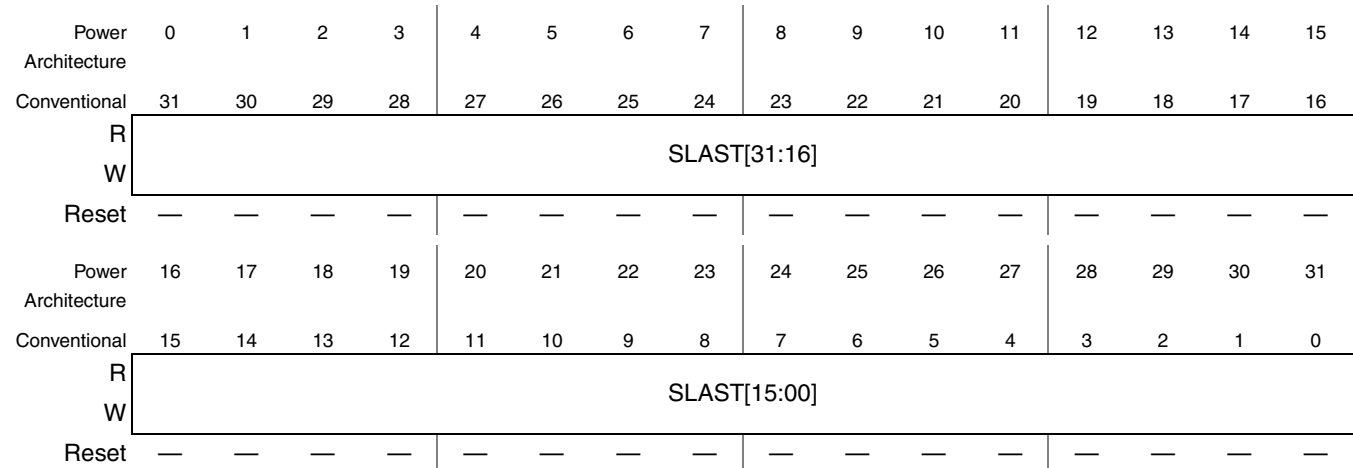


Figure 11-21. TCDn Word 3 (TCDn.slaster) Field

Table 11-24. TCDn Word 3 (TCDn.slaster) Field Descriptions

Field	Description
SLAST[31:0]	<p>Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count.</p> <p>This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.</p>

Figure 11-22 and Table 11-25 define word 4 of the TCDn structure, the daddr field.

Direct Memory Access (DMA)

DMA_Offset + 0x1000 + (32 x n) + 0x10

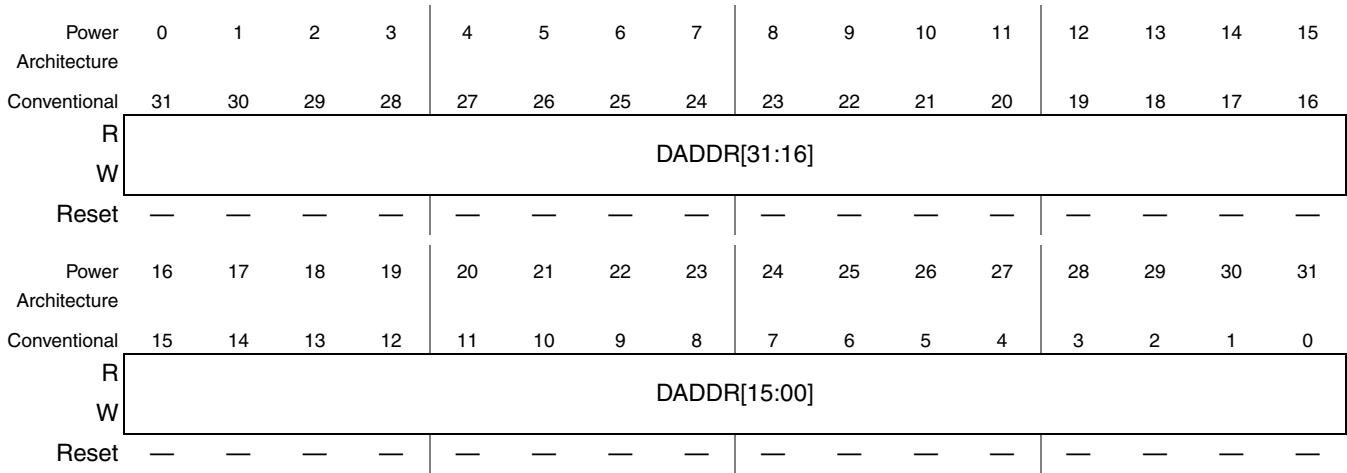


Figure 11-22. TCDn Word 4 (TCDn.daddr) Field

Table 11-25. TCDn Word 4 (TCDn.daddr) Field Descriptions

Field	Description
DADDR[31:0]	Destination address. Memory address pointing to the destination data.

Figure 11-23 and Table 11-26 define word five of the TCDn structure, the CITER and DOFF fields.

DMA_Offset + 0x1000 + (32 x n) + 0x14

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CITER.E_LINK				CITER[14:9] or CITER.LINKCH[5:0]								CITER[8:0]			
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DOFF[15:0]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 11-23. TCDn Word 5 (TCDn.{citer,doff}) Field

Table 11-26. TCDn Word 5 (TCDn.{citer,doff}) Field Descriptions

Field	Description
CITER.E_LINK	Enable Channel-to-Channel Linking on Minor Loop Complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the citer value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. This bit must be equal to the BITER.E_LINK bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
CITER[14:9] or CITER.LINKCH[5:0]	Current Major Iteration Count or Link Channel Number If TCD.CITER.E_LINK equals 0, No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field. or After the minor loop is exhausted, the DMA_ENGINE initiates a channel service request at the channel defined by CITER.LINKCH[5:0] by setting that channel's TCD.start bit. The value contained in citer.linkch[5:0] must not exceed the number of implemented channels.
CITER[8:0]	Current major iteration count
DOFF[15:0]	Destination address signed offset

Figure 11-24 and Table 11-27 define word six of the TCDn structure, the DLAST_SGA field.

$\text{DMA_Offset} + 0\text{x}1000 + (32 \times n) + 0\text{x}18$

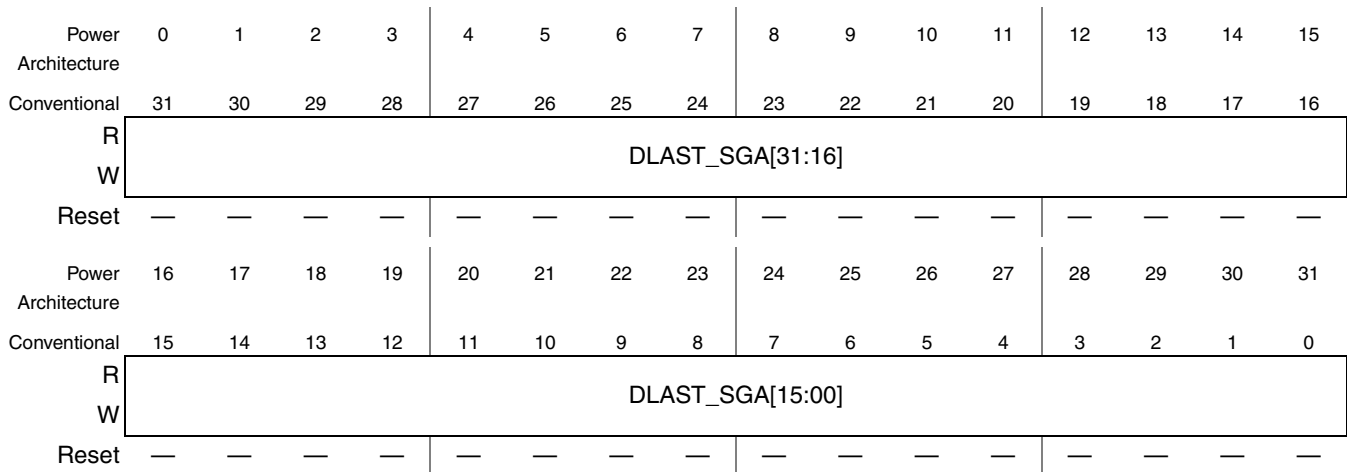


Figure 11-24. TCDn Word 6 (TCDn.dlast_sga) Field

Table 11-27. TCDn Word 6 (TCDn.dlast_sga) Field Descriptions

Field	Description
DLAST_SGA[31:0]	<p>Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather)</p> <p>If TCD.e_sg equals 0,</p> <p>Adjustment value is added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value or adjust the address to reference the next data structure.</p> <p>or1</p> <p>This address points to the beginning of a 0-modulo-32 region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32 or a configuration error is reported.</p>

Figure 11-25 and Table 11-28 define word seven of the TCDn structure, the BITER and CONTROL/STATUS fields.

DMA_Offset + 0x1000 + (32 x n) + 0x1c

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R W	BITE R. E_LI NK	BITER[14:9] OR BITER.LINKCH[5:0]							BITER[8:0]								
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R W	BWC		MAJOR.LINKCH[5:0]							DON E	ACTI VE	MAJ OR. E_LI NK	E_SG	D_RE Q	INT_ HALF	INT_ MAJ	STAR T
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	

**Figure 11-25. TCDn Word 7 (TCDn.{biter,control/status}) Fields
(Register is repeated for reference.)**

Table 11-28. TCDn Word 7 (TCDn.{biter,control/status}) Field Descriptions (Sheet 1 of 4)

Field	Description
BITER.E_LINK	Enable channel-to-channel linking on major loop complete. This is the initial value copied into the CITER.E_LINK field when the major loop is completed. The CITER.E_LINK field controls channel linking during channel execution. This bit must be equal to the CITER.E_LINK bit otherwise a configuration error is reported. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
BITER[14:9] or BITER.LINKCH[5:0]	Beginning major iteration count or beginning link channel number. This is the initial value copied into the citer field or CITER.LINKCH field when the major loop is completed. The CITER fields controls the iteration count and linking during channel execution. If TCD.BITER.E_LINK equals 0, No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit biter field. or After the minor loop is exhausted, the DMA_ENGINE initiates a channel service request at the channel defined by BITER.LINKCH[5:0] by setting that channel's TCD.START bit. The value contained in BITER.LINKCH[5:0] must not exceed the number of implemented channels.

Direct Memory Access (DMA)

DMA_Offset + 0x1000 + (32 x n) + 0x1c

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BITE R. E_LI NK	BITER[14:9] OR BITER.LINKCH[5:0]							BITER[8:0]							
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BWC	MAJOR.LINKCH[5:0]							DON E	ACTI VE	MAJ OR. E_LI NK	E_SG	D_RE Q	INT_ HALF	INT_ MAJ	STAR T
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 11-25. TCDn Word 7 (TCDn.{biter,control/status}) Fields
(Register is repeated for reference.)

Table 11-28. TCDn Word 7 (TCDn.{biter,control/status}) Field Descriptions (Sheet 2 of 4)

Field	Description
BITER[8:0]	<p>Beginning major iteration count. This is the initial value copied into the CITER field or ciTer.LINKCH field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution.</p> <p>This 9- or 15-bit count represents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16-bit BITER entry is reloaded into the 16-bit CITER entry.</p> <p>When the BITER field is initially loaded by software, it must be set to the same value as that contained in the citer field.</p> <p>If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>
BWC[1:0]	<p>Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the DMA. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, etc. sequences until the minor count is exhausted. This field forces the DMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the platform's cross-bar arbitration switch. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop.</p> <p>The dynamic priority elevation setting elevates the priority of the DMA as seen by the cross-bar arbitration switch for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles.</p> <p>00 No DMA_ENGINE stalls 01 Dynamic priority elevation 10 DMA_ENGINE stalls for four cycles after each r/w 11 DMA_ENGINE stalls for eight cycles after each r/w</p>

$$\text{DMA_Offset} + 0x1000 + (32 \times n) + 0x1c$$

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R W	BITE R. E_LINK	BITER[14:9] OR BITER.LINKCH[5:0]							BITER[8:0]								
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R W	BWC		MAJOR.LINKCH[5:0]							DONE	ACTIVE	MAJOR. E_LINK	E_SG	DREQ	INT_HALF	INT_MAJ	START
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	

Figure 11-25. TCDn Word 7 (TCDn.{biter,control/status}) Fields
(Register is repeated for reference.)

Table 11-28. TCDn Word 7 (TCDn.{biter,control/status}) Field Descriptions (Sheet 3 of 4)

Field	Description
MAJOR.LINKCH[5:0]	Link channel number If TCD.MAJOR.E_LINK equals 0, No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. or After the major loop counter is exhausted, the DMA_ENGINE initiates a channel service request at the channel defined by MAJOR.LINKCH[5:0] by setting that channel's TCD.START bit. The value contained in MAJOR.LINKCH[5:0] must not exceed the number of implemented channels.
DONE	Channel done. This flag indicates the DMA has completed the outer major loop. It is set by the DMA_ENGINE as the citer count reaches zero; it is cleared by software or the hardware when the channel is activated. This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA_ENGINE as the inner minor loop completes or if any error condition is detected.
MAJOR.E_LINK	Enable channel-to-channel linking on major loop complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
E_SG	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the DMA_ENGINE uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.

Direct Memory Access (DMA)

DMA_Offset + 0x1000 + (32 x n) + 0x1c

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BITE R. E_LI NK	BITER[14:9] OR BITER.LINKCH[5:0]							BITER[8:0]							
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BWC		MAJOR.LINKCH[5:0]						DON E	ACTI VE	MAJ OR. E_LI NK	E_SG	D_RE Q	INT_ HALF	INT_ MAJ	STAR T
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 11-25. TCDn Word 7 (TCDn.{biter,control/status}) Fields
(Register is repeated for reference.)

Table 11-28. TCDn Word 7 (TCDn.{biter,control/status}) Field Descriptions (Sheet 4 of 4)

Field	Description
D_REQ	Disable request. If this flag is set, the DMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero. 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the outer major loop is complete.
INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the DMA_ENGINE is (citer == (biter >> 1)). This halfway point interrupt request is provided to support double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when BITER values are less than two. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
START	Channel start. If this flag is set, the channel is requesting service. The DMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

11.3 Initialization/Application Information

11.3.1 DMA Initialization

A typical initialization of the DMA would be:

1. Write the DMACR register if a configuration other than the default is desired
2. Write the channel priority levels into the DCHPRIn registers if a configuration other than the default is desired
3. Enable error interrupts in the DMAEEI registers if so desired
4. Write the 32 byte TCD for each channel that may request service
5. Enable any hardware service requests via the DMAERQ register
6. Request channel service by either software (setting the TCD.START bit) or by hardware (slave device asserting its IPD_REQ signal)

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA_ENGINE reads the entire TCD for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA_ENGINE'S local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, i.e. interrupts, major loop channel linking, and scatter/gather operations, if enabled.

11.3.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors, group priority error (GPE) and channel priority error (CPE) in the DMAES register.

For all error types other than group or channel priority errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

The typical application enables error interrupts for all channels. You receive an error interrupt, but the channel number for the DMAERR register and the error interrupt request line may be wrong because they reflect the selected channel.

Channel priority errors are identified within a group after that group has been selected as the active group. For example:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group3 is the highest priority and all channels are unique in that group.
3. Group2 is the next highest priority and has two channels with the same priority level.
4. If Group3 has any service requests, those requests are executed.

5. After all of Group3 requests have completed, Group2 is the next active group.
6. If Group2 has a service request, an undefined channel in Group2 is selected and a channel priority error occurs.
7. This repeats until the all of the Group2 requests have been removed or a higher priority Group3 request comes in.

A group priority error is global and any request in any group causes a group priority error.

In general, if priority levels are not unique, the highest (channel/group) priority with an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA_ENGINE. The hardware service request handshake signals, error interrupts, and error reporting is associated with the selected channel.

11.3.3 DMA Arbitration Mode Considerations

11.3.3.1 Fixed Group Arbitration, Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the DMA is programmed so the channels within one group use fixed priorities and that group is assigned the highest fixed priority of all groups, that group may take all the bandwidth of the DMA controller. No other groups are serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request.

The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

11.3.3.2 Round Robin Group Arbitration, Fixed Channel Arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round robin algorithm selects the highest pending request from the next group in the round robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence or skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, that channel is always serviced before lower priority channels in the same group. Therefore, the lower priority channels are never serviced.

The advantage of this scenario is that no one group consumes all the DMA bandwidth.

The highest priority channel selection latency is potentially greater than fixed/fixed arbitration.

Excessive request rates on high priority channels could prevent the servicing of lower priority channels in the same group.

11.3.3.3 Round Robin Group Arbitration, Round Robin Channel Arbitration

Groups are serviced as described in section [Section 11.3.3.2, “Round Robin Group Arbitration, Fixed Channel Arbitration,”](#) but channels are serviced in channel number order this time. Only one channel is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round robin manner, any channel that generates DMA requests faster than a combination of the group round robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests not serviced are simply lost, but at least one channel is serviced.

This scenario ensures all channels are guaranteed service at some point, regardless of the request rates. However, the potential latency could be quite high.

All channels are treated equally. Priority levels are not used in round robin mode.

11.3.3.4 Fixed Group Arbitration, Round Robin Channel Arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in section [Section 11.3.3.1, “Fixed Group Arbitration, Fixed Channel Arbitration,”](#) but all the channels in the highest priority group are serviced.

Service latency is short on the highest priority group, but could potentially become longer as the group priority decreases.

11.3.4 DMA Transfer

11.3.4.1 Single Request

To perform a single transfer of n bytes of data with one activation, set the major loop to one ($\text{TCD.CITER} = \text{TCD.BITER} = 1$). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit is set and an interrupt is generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The DMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

```
TCD.citer = TCD.biter = 1
TCD.nbytes = 16
TCD.saddr = 0x1000
TCD.soff = 1
TCD.ssize = 0
TCD.slast = -16
TCD.daddr = 0x2000
TCD.doff = 4
TCD.dsize = 2
TCD.dlast_sga = -16
TCD.int_maj = 1
TCD.start = 1          (TCD.word7 should be written last after all other fields have
                        been initialized)

All other TCD fields = 0
```

These settings generate the following sequence of events:

1. IPS write to the TCD.start bit requests channel service
2. The channel is selected by arbitration for servicing
3. DMA_ENGINE writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1
4. DMA_ENGINE reads: channel TCD data from local memory to internal register file
5. The source to destination transfers are executed as follows:
 - a) READ_BYTE(0X1000), READ_BYTE(0X1001), READ_BYTE(0X1002), READ_BYTE(0X1003)
 - b) WRITE_WORD(0x2000) → first iteration of the minor loop
 - c) READ_BYTE(0X1004), READ_BYTE(0X1005), READ_BYTE(0X1006), READ_BYTE(0X1007)
 - d) WRITE_WORD(0x2004) → second iteration of the minor loop
 - e) READ_BYTE(0X1008), READ_BYTE(0X1009), READ_BYTE(0X100A), READ_BYTE(0X100B)
 - f) WRITE_WORD(0x2008) → third iteration of the minor loop
 - g) READ_BYTE(0X100C), READ_BYTE(0X100D), READ_BYTE(0X100E), READ_BYTE(0X100F)
 - h) WRITE_WORD(0x200c) → last iteration of the minor loop → major loop complete
6. DMA_ENGINE writes: TCD.SADDR = 0X1000, TCD.DADDR = 0X2000, TCD.CITER = 1 (TCD.BITER)
7. DMA_ENGINE writes: TCD.ACTIVE = 0, TCD.DONE = 1, DMAINT[n] = 1
8. The channel retires

The DMA becomes idle or services next channel.

11.3.4.2 Multiple Requests

The next example is the same as previous with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The DMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the DMAERQ register, channel service requests are initiated by the slave device.

```
TCD.citer = TCD.biter = 2
TCD.slast = -32
TCD.dlast_sga= -32
```

These settings generate the following sequence of events:

1. First hardware (IPD_REQ) request for channel service
2. The channel is selected by arbitration for servicing
3. DMA_ENGINE writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
4. DMA_ENGINE reads: channel TCD data from local memory to internal register file
5. The source to destination transfers are executed as follows:
 - a) READ_BYTE(0X1000), READ_BYTE(0X1001), READ_BYTE(0X1002), READ_BYTE(0X1003)
 - b) WRITE_WORD(0x2000) → first iteration of the minor loop
 - c) READ_BYTE(0X1004), READ_BYTE(0X1005), READ_BYTE(0X1006), READ_BYTE(0X1007)
 - d) WRITE_WORD(0x2004) → second iteration of the minor loop
 - e) READ_BYTE(0X1008), READ_BYTE(0X1009), READ_BYTE(0X100A), READ_BYTE(0X100B)
 - f) WRITE_WORD(0x2008) → third iteration of the minor loop
 - g) READ_BYTE(0X100C), READ_BYTE(0X100D), READ_BYTE(0X100E), READ_BYTE(0X100F)
 - h) WRITE_WORD(0x200c) → last iteration of the minor loop
6. DMA_ENGINE writes: TCD.SADDR = 0X1010, TCD.DADDR = 0X2010, TCD.CITER = 1
7. DMA_ENGINE writes: TCD.ACTIVE = 0
8. The channel retires → one iteration of the major loop

The DMA goes idle or services next channel.

9. Second hardware (IPD_REQ) requests channel service
10. The channel is selected by arbitration for servicing
11. DMA_ENGINE writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1
12. DMA_ENGINE reads: channel TCD data from local memory to internal register file,
13. The source to destination transfers are executed as follows:
 - a) READ_BYTE(0X1010), READ_BYTE(0X1011), READ_BYTE(0X1012), READ_BYTE(0X1013)
 - b) WRITE_WORD(0x2010) → first iteration of the minor loop

- c) READ_BYTE(0X1014), READ_BYTE(0X1015), READ_BYTE(0X1016),
READ_BYTE(0X1017)
 - d) WRITE_WORD(0x2014) → second iteration of the minor loop
 - e) READ_BYTE(0X1018), READ_BYTE(0X1019), READ_BYTE(0X101A),
READ_BYTE(0X101B)
 - f) WRITE_WORD(0x2018) → third iteration of the minor loop
 - g) READ_BYTE(0X101C), READ_BYTE(0X101D), READ_BYTE(0X101E),
READ_BYTE(0X101F)
 - h) WRITE_WORD(0x201c) → last iteration of the minor loop → major loop complete
14. DMA_ENGINE writes: TCD.SADDR = 0X1000, TCD.DADDR = 0X2000, TCD.CITER = 2
(TCD.BITER)
15. DMA_ENGINE writes: TCD.ACTIVE = 0, TCD.DONE = 1, DMAINT[N] = 1
16. The channel retires → major loop complete

The DMA becomes idle or services the next channel.

11.3.5 TCD Status

11.3.5.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the TCD.START bit and the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a one. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel completed minor loop and is idle)
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel completed major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. IPD_REQ asserts (channel service request via hardware)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel completed minor loop and is idle)
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel completed major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

11.3.5.2 Active Channel TCD Reads

The DMA reads back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the DMA_ENGINE is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

11.3.5.3 Preemption Status

Preemption is only available when fixed arbitration is selected for group and channel arbitration modes. A preemptable situation is when a preempt-enabled channel is running and a higher priority request becomes active. When the DMA_ENGINE is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

The worst case latency when switching to a preempt channel is the summation of:

- Arbitration latency (2 cycles)
- Bandwidth control stalls (if enabled)
- The time to execute two read/write sequences (including AHB bus holds; a system dependency driven by the slave devices or the crossbar)

11.3.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.START bit of another channel (or itself) that initiates a service request for that channel. This operation is automatically performed by the DMA_ENGINE at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field is used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made.

For example, with the initial fields of:

```
TCD.citer.e_link= 1
TCD.citer.linkch= 0xC
TCD.citer.value= 0x4
TCD.major.e_link= 1
TCD.major.linkch= 0x7
```

Execute as:

1. Minor loop done → set channel 12 TCD.start bit
2. Minor loop done → set channel 12 TCD.start bit
3. Minor loop done → set channel 12 TCD.start bit
4. Minor loop done, major loop done → set channel 7 TCD.start bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a 9-bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the citer value to increase the range of the citer.

NOTE

The TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must equal or a configuration error is reported. The citer and biter vector widths must be equal to calculate the major loop, half-way done interrupt point.

11.3.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

11.3.7.1 Dynamic Priority Changing

The following two options are recommended for dynamically changing channel priority levels:

- Switch to round-robin channel arbitration mode, change the channel priorities, and then switch back to fixed arbitration mode
- Disable all the channels within a group, change the channel priorities within that group only, and then enable the appropriate channels.

The following two options are available for dynamically changing group priority levels:

- Switch to round-robin group arbitration mode, change the group priorities, and then switch back to fixed arbitration mode,
- Disable all channels, change the group priorities, and then enable the appropriate channels.

11.3.7.2 Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD

local memory at the end of channel execution, allowing you to enable either feature during channel execution.

Because you can change the configuration during execution, a coherency model is needed. Consider the scenario where you attempt to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the DMA_ENGINE is retiring the channel. The TCD.MAJOR.E_LINK would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.MAJOR.E_LINK bit.
2. Read back the TCD.MAJOR.E_LINK bit.
3. Test the TCD.MAJOR.E_LINK request status.
4. If the bit is set, the dynamic link attempt was successful.
5. If the bit is cleared, the attempted dynamic link did not succeed. The channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD.WORD7 after that channel's TCD.DONE bit is set, indicating the major loop is complete.

NOTE

Clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the DMA_ENGINE after a channel begins execution.

Chapter 12

Display Interface Unit (DIU)

12.1 Introduction

The DIU is a display controller designed to manage TFT LCD display. Besides generating all the signals required to drive the display, the DIU manages real-time blending of up to three planes onto the display.

12.1.1 Features

- Display color depth: up to 24 bpp
- Display interfaces: parallel TTL
- Maximum number of physical input planes: 3
 - Memory write-back mode to store intermediate results, extending the number of graphics planes
- Input pixel formats: RGB and 256-level grayscale
- Programmable bit order definition up to 8 bits per component
- Hardware cursor: 32x32 pixels, 16 bpp
- α -blending range: up to 256 levels
- Chroma Keying: Selectable by range
- Independent programmable gamma adjustments for each color component

12.1.2 Modes of Operation

The DIU has five modes of operation:

Mode 0: DIU OFF. In this mode, the DIU is disabled.

Mode 1: All three planes output to display. This is the typical operating mode of the DIU.

Mode 2: Plane 1 to display, Planes 2 and 3 written back to memory. This mode is used to display a plane while processing (and writing back to memory) the data for other planes.

Mode 3: All three planes written back to memory. This mode is used to process (and write back to memory) the data for the planes without displaying any of them.

Mode 4: Color Bar Generation. This is a debug mode to check the operation of the DIU without the need for setting up the display memory structures in memory.

These modes are set by programming the DIU_MODE register. See [Section 12.3.3.8, “DIU_MODE Register](#) for more information.

12.2 External Signal Description

Table 12-2 describes the DIU input and output signals, the meaning of their different states, and relative timing information for assertion and negation.

Table 12-2. Display Interface Detailed Signal Descriptions

Signal	I/O	Description
DIU_CLK ¹	O	Pixel clock. This signals is used to drive the display panel.
DIU_VSYNC	O	Vertical synchronizing signal. This signal indicates the beginning of a new frame. This signal may alternately be programmed to output a composite sync (CSYNC) signal by programing SYN_POL[BP_VS]. See Section 12.3.3.16, “SYN_POL Register for more information. The composite sync signal combines the horizontal and vertical synchronizing signals to form a composite synchronizing signal. It includes both the HSYNC pulse and the VSYNC pulse. The default output is DIU_VSYNC.
		State Meaning Asserted at the beginning of a new frame.
		Timing Asserted with the first cycle of the frame period. The length of the pulse is programmable.
DIU_HSYNC	O	Horizontal synchronizing signal. This signal indicates the beginning of a new line. This signal may alternately be programmed to output a composite sync (CSYNC) signal by programing SYN_POL[BP_HS]. See Section 12.3.3.16, “SYN_POL Register for more information. The composite sync signal combines the horizontal and vertical synchronizing signals to form a composite synchronizing signal. It includes both the HSYNC pulse and the VSYNC pulse. The default output is DIU_HSYNC.
		State Meaning Asserted at the beginning of a new line.
		Timing Asserted with the first cycle of a new line. The length of the pulse is programable.
DIU_DE	O	Data enable. This signal qualifies the data on the data output signals (DIU_LD)
		State Meaning Deasserted: DIU_LD data is not valid Asserted: DIU_LD data is valid.
DIU_LD[23:0]	O	Data output signals. <ul style="list-style-type: none"> DIU_LD[23:16] = Red[7:0]. DIU_LD[23] is the most significant bit, and DIU_LD[16] is the least significant bit of the Red component. DIU_LD[15:8] = Green[7:0]. DIU_LD[15] is the most significant bit, and DIU_LD[8] is the least significant bit of the Green component. DIU_LD[7:0] = Blue[7:0]. DIU_LD[7] is the most significant bit, and DIU_LD[0] is the least significant bit of the Blue component.

¹ Refer to the system clock chapter for details on this clock.

12.3 Memory Map and Register Definition

12.3.1 Memory Map

Table 12-3 shows the register memory map for the DIU memory controller.

Table 12-3. DIU Memory Map

Offset	Register	Access	Section/Page
0x00	DESC_1 — Pointer to the Area Descriptor of Plane1	R/W	12.3.3.1/12-8
0x04	DESC_2 — Pointer to the Area Descriptor of Plane2	R/W	12.3.3.2/12-9
0x08	DESC_3 — Pointer to the Area Descriptor of Plane3	R/W	12.3.3.3/12-10
0x0c	GAMMA — Pointer to Gamma Table	R/W	12.3.3.4/12-11
0x10	PALETTE — Pointer to Palette	R/W	12.3.3.5/12-12
0x14	CURSOR — Pointer to Cursor Bitmap	R/W	12.3.3.6/12-13
0x18	CURS_POS — Position of the cursor in the display	R/W	12.3.3.7/12-13
0x1c	DIU_MODE — DIU Mode of Operation	R/W	12.3.3.8/12-15
0x20	BGND — Background Color	R/W	12.3.3.9/12-16
0x24	BGND_WB — Background Color in write back Mode	R/W	12.3.3.10/12-17
0x28	DISP_SIZE — Display Size	R/W	12.3.3.11/12-18
0x2c	WB_SIZE — Write back Plane Size	R/W	12.3.3.12/12-18
0x30	WB_MEM_ADDR — Address to Store the write back Plane	R/W	12.3.3.13/12-19
0x34	HSYN_PARA — Horizontal synchronization pulse parameters	R/W	12.3.3.14/12-20
0x38	VSYN_PARA — Vertical synchronization pulse parameters	R/W	12.3.3.15/12-20
0x3c	SYN_POL — Synchronization Signals Polarity	R/W	12.3.3.16/12-22
0x40	THRESHOLDS — The Thresholds	R/W	12.3.3.17/12-23
0x44	INT_STATUS — Interrupt Status Register	R	12.3.3.18/12-24
0x48	INT_MASK — Interrupt Mask Register	R/W	12.3.3.19/12-25
0x4c	COLORBAR_1 — Color #1 in the Color Bar, Black	R/W	12.3.3.20/12-25
0x50	COLORBAR_2 — Color #2 in the Color Bar, Blue	R/W	12.3.3.20/12-25
0x54	COLORBAR_3 — Color #3 in the Color Bar, Cyan	R/W	12.3.3.20/12-25
0x58	COLORBAR_4 — Color #4 in the Color Bar, Green	R/W	12.3.3.20/12-25
0x5c	COLORBAR_5 — Color #5 in the Color Bar, Yellow	R/W	12.3.3.20/12-25
0x60	COLORBAR_6 — Color #6 in the Color Bar, Red	R/W	12.3.3.20/12-25
0x64	COLORBAR_7 — Color #7 in the Color Bar, Purple	R/W	12.3.3.20/12-25
0x68	COLORBAR_8 — Color #8 in the Color Bar, White	R/W	12.3.3.20/12-25

Table 12-3. DIU Memory Map (continued)

Offset	Register	Access	Section/Page
0x6c	FILLING — Input, output buffer filling status, for debug purpose	R	12.3.3.21/12-30
0x70	PLUT — Priority Look Up Table	R/W	12.3.3.22/12-31

12.3.2 Register Summary

Table 12-4. DIU Block Register Summary (Sheet 1 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DESC_1 0x00	R	DESC_1[31:16]															
	W																
	R	DESC_1[15: 0]															
	W																
DESC_2 0x04	R	DESC_2[31:16]															
	W																
	R	DESC_2[15: 0]															
	W																
DESC_3 0x08	R	DESC_3[31:16]															
	W																
	R	DESC_3[15: 0]															
	W																
GAMMA 0x0c	R	GAMMA[31:16]															
	W																
	R	GAMMA[15: 0]															
	W																
PALETTE 0x10	R	PALETTE[31:16]															
	W																
	R	PALETTE[15: 0]															
	W																
CURSOR 0x14	R	CURSOR[31:16]															
	W																
	R	CURSOR[15: 0]															
	W																

Table 12-4. DIU Block Register Summary (Sheet 2 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CURS_POS 0x18	R	0	0	0	0	0	0	CURSOR_Y[10: 0]									
	W																
	R	0	0	0	0	0	0	CURSOR_X[10: 0]									
	W																
DIU_MODE 0x1c	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	DIU_MODE[2:0]			
	W																
BGND 0x20	R	0	0	0	0	0	0	0	0	BGND_R[7:0]							
	W																
	R	BGND_G[7:0]								BGND_B[7:0]							
	W																
BGND_WB 0x24	R	0	0	0	0	0	0	0	0	BGND_WB_R[7:0]							
	W																
	R	BGND_WB_G[7:0]								BGND_WB_B[7:0]							
	W																
DISP_SIZE 0x28	R	0	0	0	0	0	DELTA_Y[10: 0]										
	W																
	R	0	0	0	0	0	DELTA_X[10: 0]										
	W																
WB_SIZE 0x2c	R	0	0	0	0	0	DELTA_Y_WB[10: 0]										
	W																
	R	0	0	0	0	0	DELTA_X_WB[10: 0]										
	W																
WB_MEM_AD DR 0x30	R	WB_MEM_ADDR[31:16]															
	W																
	R	WB_MEM_ADDR[15: 0]															
	W																
HSYN_PARA 0x34	R	BP_H[9:0]										0	PW_H[9:5]				
	W																
	R	PW_H[4:0]					0	FP_H[9:0]									
	W																

Table 12-4. DIU Block Register Summary (Sheet 3 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
VSYN_PARA 0x38	R	BP_V[9:0]										0	PW_V[9:5]					
	W																	
	R	PW_V[4:0]					0	FP_V[9:0]										
	W																	
SYN_POL 0x3c	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																	
	R	0	0	0	0	0	0	0	0	0	0	0	BP_VS	BP_HS	INV_CS	INV_VS	INV_HS	
	W																	
THRESHOLDS 0x40	R	1	0	0	0	0	LS_BF_VS[10:0]											
	W																	
	R	1	1	1	1	1	0	0	0	OUT_BUF_LOW[7:0]								
	W																	
INT_STATUS 0x44	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																	
	R	0	0	0	0	0	0	0	0	0	0	WB_PEND	LS_BF_VS	PARERR	UNDRUN	VSYNC_WB	VSYNC	
	W																	
INT_MASK 0x48	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	W																	
	R	0	0	0	0	0	0	0	0	0	0	M_WB_PEND	M_LS_BF_VS	M_PARERR	M_UNDRUN	M_VSYN_WB	M_VSYNC	
	W																	
COLBAR_1 0x4c	R	1	1	1	1	1	1	1	1	COLBAR_1_R[7:0]								
	W																	
	R	COLBAR_1_G[7:0]								COLBAR_1_B[7:0]								
	W																	
COLBAR_2 0x50	R	1	1	1	1	1	1	1	1	COLBAR_2_R[7:0]								
	W																	
	R	COLBAR_2_G[7:0]								COLBAR_2_B[7:0]								
	W																	

Table 12-4. DIU Block Register Summary (Sheet 4 of 4)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COLBAR_3 0x54	R	1	1	1	1	1	1	1	1	COLBAR_3_R[7:0]							
	W																
	R	COLBAR_3_G[7:0]								COLBAR_3_B[7:0]							
	W																
COLBAR_4 0x58	R	1	1	1	1	1	1	1	1	COLBAR_4_R[7:0]							
	W																
	R	COLBAR_4_G[7:0]								COLBAR_4_B[7:0]							
	W																
COLBAR_5 0x5c	R	1	1	1	1	1	1	1	1	COLBAR_5_R[7:0]							
	W																
	R	COLBAR_5_G[7:0]								COLBAR_5_B[7:0]							
	W																
COLBAR_6 0x60	R	1	1	1	1	1	1	1	1	COLBAR_6_R[7:0]							
	W																
	R	COLBAR_6_G[7:0]								COLBAR_6_B[7:0]							
	W																
COLBAR_7 0x64	R	1	1	1	1	1	1	1	1	COLBAR_7_R[7:0]							
	W																
	R	COLBAR_7_G[7:0]								COLBAR_7_B[7:0]							
	W																
COLBAR_8 0x68	R	1	1	1	1	1	1	1	1	COLBAR_8_R[7:0]							
	W																
	R	COLBAR_8_G[7:0]								COLBAR_8_B[7:0]							
	W																
FILLING 0x6c	R	0	0	0	0	0	0	FILLING_OBF[9:0]									
	W																
	R	FILLING_WB[3:0]				FILLING_P3[3:0]				FILLING_P2[3:0]				FILLING_P1[3:0]			
	W																
PLUT 0x70	R	PRIORITY_7[3:0]				PRIORITY_6[3:0]				PRIORITY_5[3:0]				PRIORITY_4[3:0]			
	W																
	R	PRIORITY_3[3:0]				PRIORITY_2[3:0]				PRIORITY_1[3:0]				PRIORITY_0[3:0]			
	W																

12.3.3 Register Descriptions

12.3.3.1 DESC_1 Register

Offset 0x00Access: User read/write

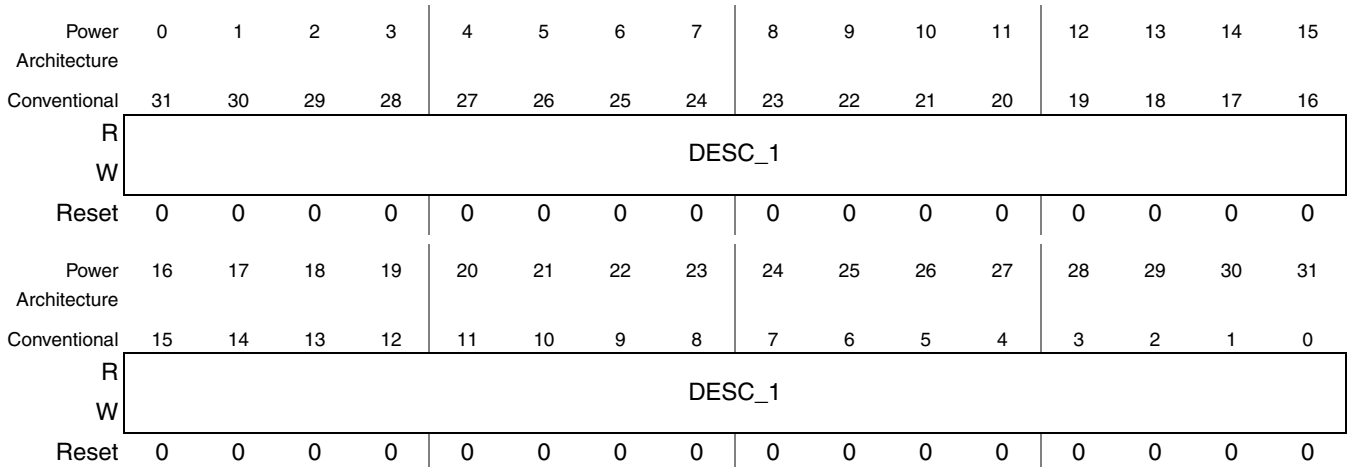


Figure 12-1. Plane 1 Area Descriptor Pointer Register (DESC_1)

Table 12-5. DESC_1 Field Descriptions

Field	Description
DESC_1	DESC_1 register is the plane one area descriptor pointer. It sets the base address of the first plane one AD (Area Descriptor). This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). DESC_1 = 0x0000_0000 means no AD available for this plane.

12.3.3.2 DESC_2 Register

Offset 0x04Access: User read/write

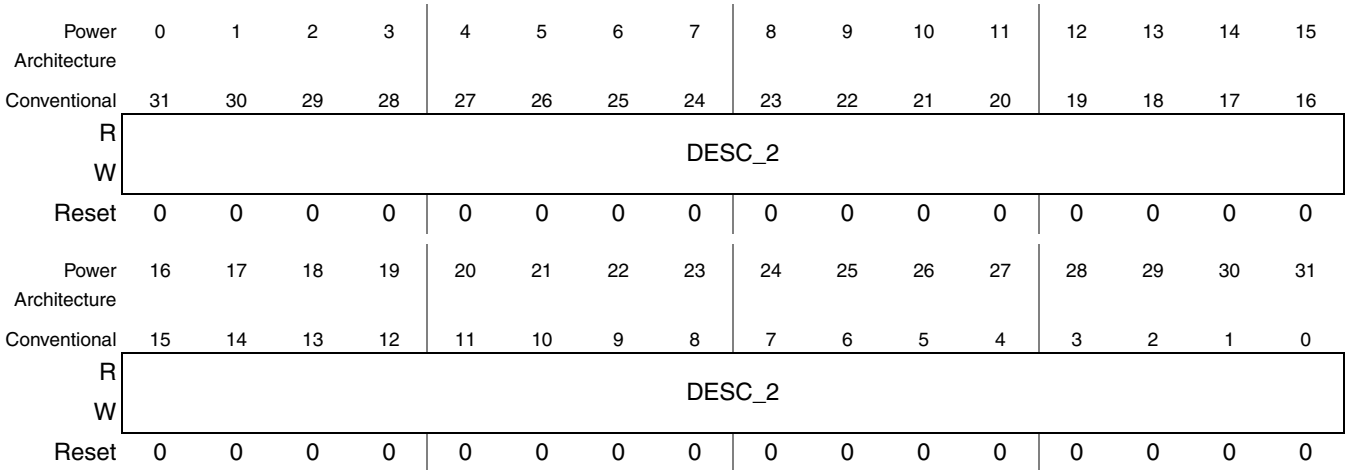


Figure 12-2. Plane 2 Area Descriptor Pointer Register (DESC_2)

Table 12-6. DESC_2 Field Descriptions

Field	Description
DESC_2	DESC_2 register is the plane two area descriptor pointer. It sets the base address of the first plane two AD. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0) .DESC_2 = 0x0000_0000 means no AD available for this plane.

12.3.3.3 DESC_3 Register

Offset 0x08Access: User read/write

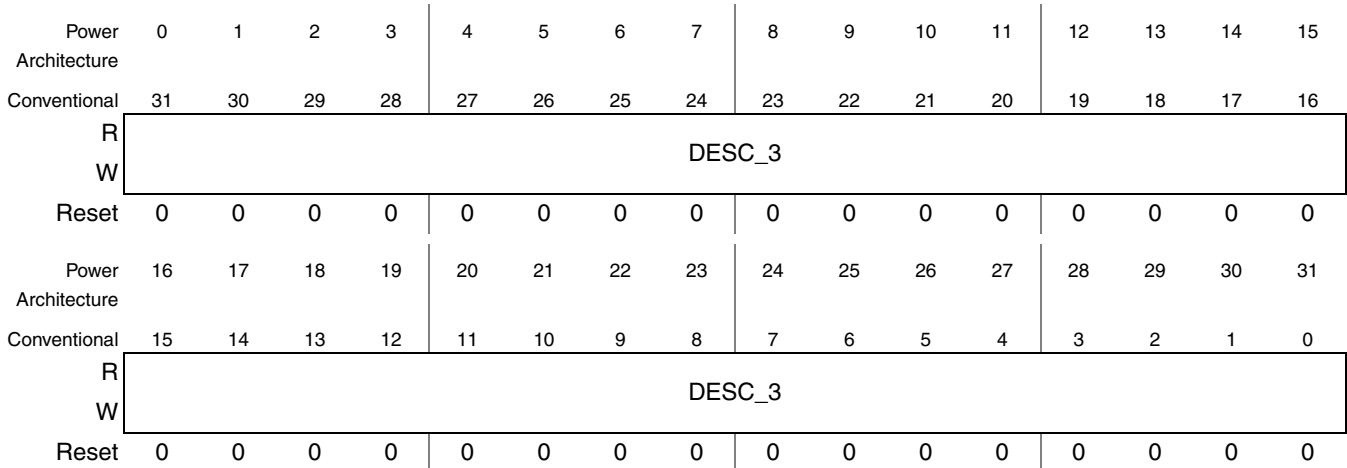


Figure 12-3. Plane 3 Area Descriptor Pointer Register (DESC_3)

Table 12-7. DESC_3 Field Descriptions

Field	Description
DESC_3	DESC_3 register is the plane three area descriptor pointer. It sets the base address of the first plane three AD. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). DESC_3 = 0x0000_0000 means no AD available for this plane.

12.3.3.4 GAMMA Register

Offset 0x0cAccess: User read/write

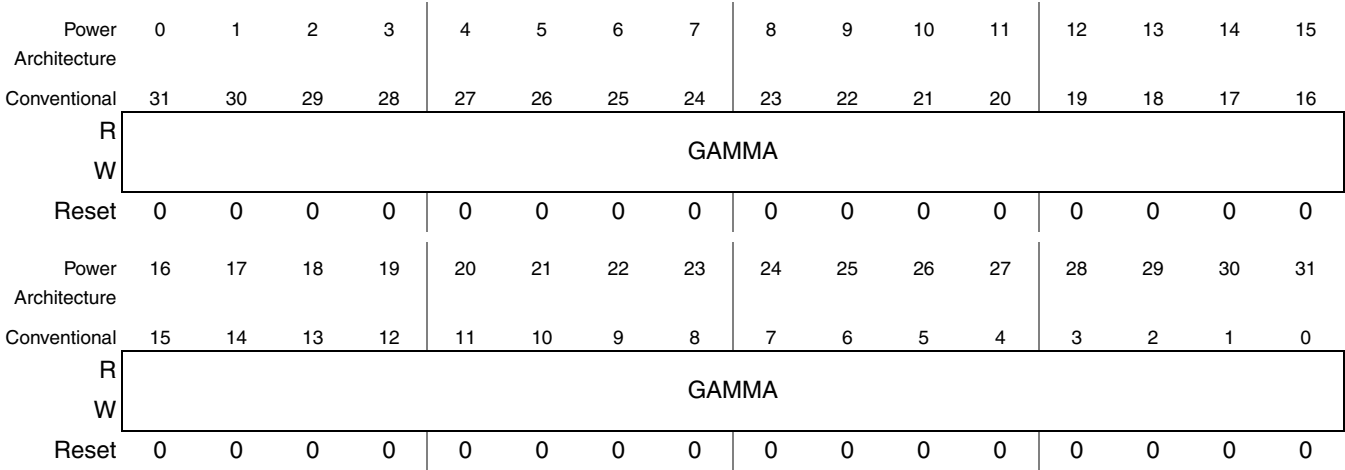


Figure 12-4. GAMMA Register

Table 12-8. GAMMA Field Descriptions

Field	Description
GAMMA	GAMMA register sets the base address to the GAMMA table in memory. Writing to this register causes the DIU to load the new GAMMA table from there. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). 32-byte boundary aligned address is more efficient.

12.3.3.5 PALETTE Register

Offset 0x10Access: User read/write

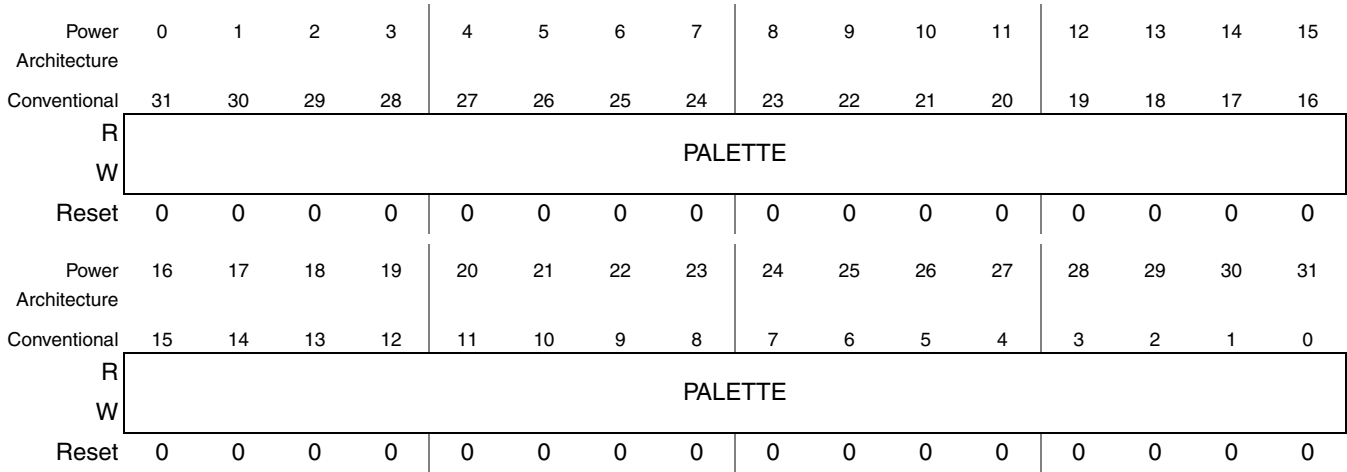


Figure 12-5. PALETTE Register

Table 12-9. PALETTE Field Descriptions

Field	Description
PALETTE	PALETTE register sets the base address to the Palette table in memory. Writing to this register causes the DIU to load the new Palette table from there. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). 32-byte boundary aligned address is more efficient.

12.3.3.6 CURSOR Register

Offset 0x14 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CURSOR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CURSOR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-6. CURSOR Register

Table 12-10. CURSOR Field Descriptions

Field	Description
CURSOR	CURSOR register sets the base address to the CURSOR bitmap in memory. Writing to this register causes the DIU to load the new CURSOR bitmap from there. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). 32-byte boundary aligned address is more efficient.

12.3.3.7 CURS_POS Register

CUR_POS register sets the position of the cursor in the display. [Table 12-11](#) shows it's field descriptions.

Display Interface Unit (DIU)

Offset 0x18Access: User read/write

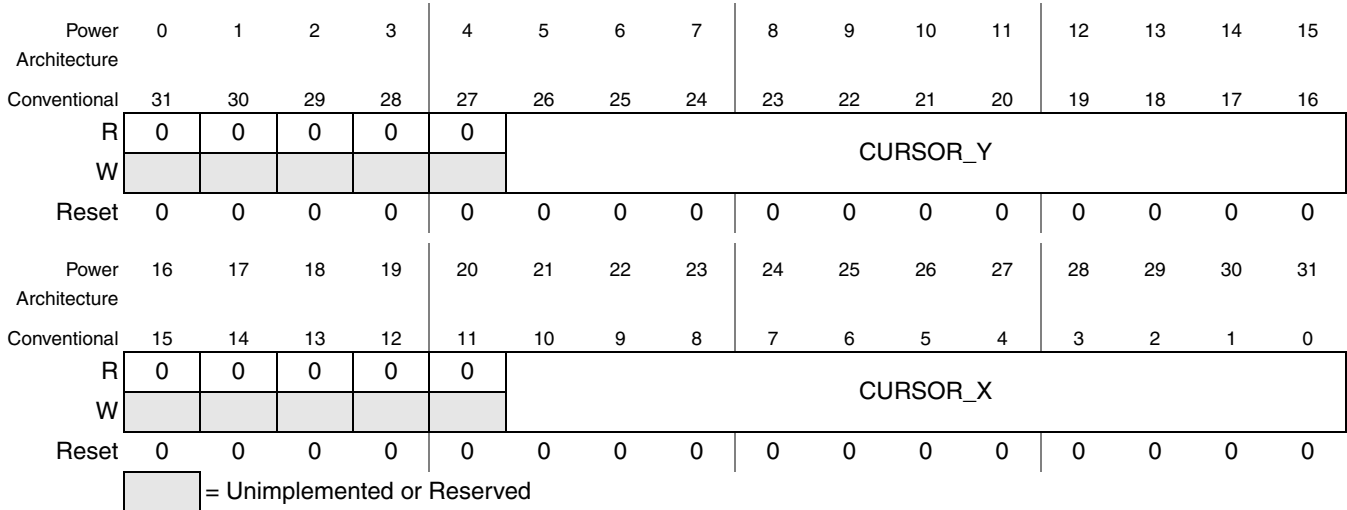


Figure 12-7. CURS_POS Register

Table 12-11. CURS_POS Field Descriptions

Field	Description
CURSOR_Y	Vertical position of the cursor (in pixels), from the top-left corner.
CURSOR_X	Horizontal position of the cursor (in pixels), from the top-left corner.

12.3.3.8 DIU_MODE Register

DIU_MODE register sets the operation mode of the DIU. See [Table 12-12](#) for its field descriptions.

Offset 0x1cAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DIU_MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 12-8. DIU_MODE Register

Table 12-12. DIU_MODE Field Descriptions

Field	Description
DIU_MODE	DIU Operation Mode 000 Encoding Mode 0: DIU OFF. 001 Encoding Mode 1: All three planes output to display. 010 Encoding Mode 2: Plane 1 to display, Planes 2 and 3 written back to memory. 011 Encoding Mode 3: All three planes written back to memory. 100 Encoding Mode 4: Color Bar Generation. All other encodings are reserved ¹

¹ Writing a reserved value is blocked and doesn't affect the register value.

12.3.3.9 BGND Register

BGND register sets the default background color for plane one (for mode 1 or 2). This is the color used to fill the areas for which no data is assigned in the area descriptor.

Offset 0x20Access: User read/write

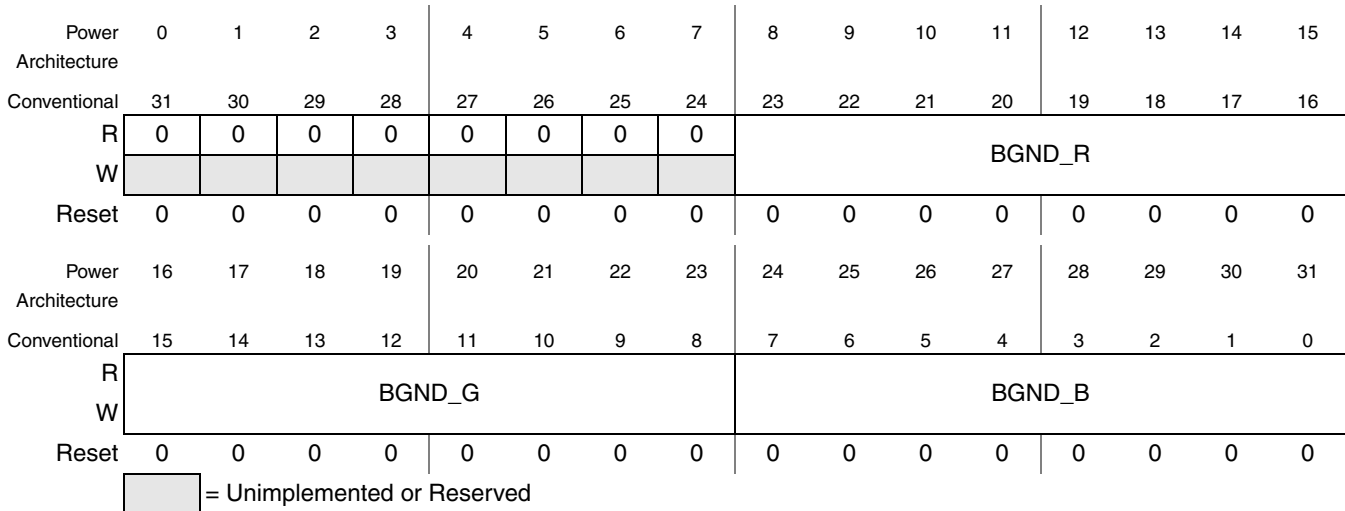


Figure 12-9. BGBD Register

Table 12-13. BGBD Field Descriptions

Field	Description
BGND_R	BGND_R represents the red component of the background.
BGND_G	BGND_G represents the green component of the background.
BGND_B	BGND_B represents the blue component of the background.

12.3.3.10 BGND_WB Register

BGND_WB sets default background color for the write back planes (plane two in mode 2 or plane one in mode 3). This is the color used to fill the areas for which no data is assigned in the area descriptor.

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	BGND_WB_R							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BGND_WB_G								BGND_WB_B							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-10. BGBD_WB Register

Table 12-14. BGBD_WB Field Descriptions

Field	Description
BGND_WB_R	BGND_WB_R represents the red component of the background for the write back planes.
BGND_WB_G	BGND_WB_G represents the green component of the background for the write back planes.
BGND_WB_B	BGND_WB_B represents the blue component of the background for the write back planes.

12.3.3.11 DISP_SIZE Register

DISP_SIZE register sets the display size (in pixels).

Offset 0x28Access: User read/write

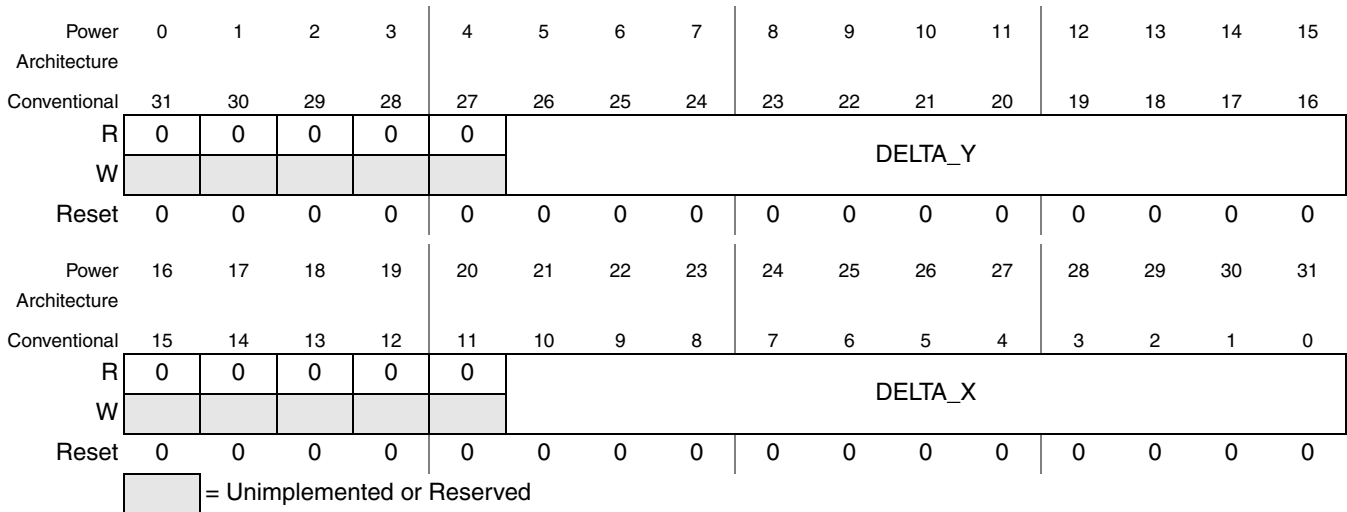


Figure 12-11. DISP_SIZE Register

Table 12-15. DISP_SIZE Field Descriptions

Field	Description
DELTA_Y	DELTA_Y represents the vertical resolution.
DELTA_X	DELTA_X represents the horizontal resolution.

12.3.3.12 WB_SIZE Register

WB_SIZE register sets the write back frame size (in pixels).

Offset 0x2cAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	DELTA_Y_WB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	DELTA_X_WB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-12. WB_SIZE Register

Table 12-16. WB_SIZE Field Descriptions

Field	Description
DELTA_Y_WB	DELTA_Y_WB represents the vertical resolution.
DELTA_X_WB	DELTA_X_WB represents the horizontal resolution.

12.3.3.13 WB_MEM_ADDR Register

Offset 0x30Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WB_MEM_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WB_MEM_ADDR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-13. WB_MEM_ADDR Register

Table 12-17. WB_MEM_ADDR Field Descriptions

Field	Description
WB_MEM_ADDR	WB_MEM_ADDR register sets the base address where the write back frame is written to in the memory. Write to this register triggers a write back frame refresh. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). 32-byte boundary aligned address is more efficient.

12.3.3.14 HSYN_PARA Register

HSYN_PARA register sets timing parameters related to the horizontal synchronization signal generation. See [Figure 12-49](#), the display timing diagrams, for detailed signal meaning.

Offset 0x34 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BP_H										0	PW_H				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PW_H					0	FP_H									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-14. HSYN_PARA Register

Table 12-18. HSYN_PARA Field Descriptions

Field	Description
BP_H	HSYNC back-porch pulse width (in pixel clock cycles). It can be 0.
PW_H	HSYNC active pulse width (in pixel clock cycles). It must be greater than or equal to 1.
FP_H	HSYNC front-porch pulse width (in pixel clock cycles). It can be 0.

12.3.3.15 VSYN_PARA Register

VSYN_PARA register sets timing parameters related to the vertical synchronization signal generation. See [Figure 12-50](#), the display timing diagram, for detailed signal meaning.

Offset 0x38 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BP_V										0	PW_V				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PW_V					0	FP_V									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-15. VSYN_PARA Register

Table 12-19. VSYN_PARA Field Descriptions

Field	Description
BP_V	VSYN back-porch pulse width (in HSYNC signal cycles) It can be 0.
PW_V	VSYN active pulse width (in HSYNC signal cycles). It must be greater than or equal to 1.
FP_V	VSYN front-porch pulse width (in HSYNC signal cycles). it can be 0.

12.3.3.16 SYN_POL Register

SYN_POL register selects polarity for corresponding synchronize signals (HSYNC, VSYNC, CSYNC) and controls the bypass of HSYNC or VSYNC with CSYNC signal.

Offset 0x3c Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	BP_VS	BP_HS	INV_CS	INV_VS	INV_HS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-16. SYN_POL Register

Table 12-20. SYN_POL Field Descriptions

Field	Description
BP_VS	Bypass Vertical Synchronize Signal (internal pin muxing) 0 Not bypass VSYNC signal output 1 CSYNC bypass VSYNC signal, output CSYNC instead of VSYNC
BP_HS	Bypass Horizontal Synchronize Signal (internal pin muxing) 0 Not bypass HSYNC signal output 1 CSYNC bypass HSYNC signal, output CSYNC instead of HSYNC
INV_CS	Invert Composite Synchronize Signal 0 Not invert CSYNC signal, active HIGH 1 Invert CSYNC signal, active LOW
INV_VS	Invert Vertical Synchronize Signal 0 Not invert VSYNC signal, active HIGH 1 Invert VSYNC signal, active LOW
INV_HS	Invert Horizontal Synchronize Signal 0 1 Not invert HSYNC signal, active HIGH 1 Invert HSYNC signal, active LOW

12.3.3.17 THRESHOLDS Register¹

THRESHOLDS register sets three useful threshold values related to DIU operations.

Offset 0x40 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	0	0	0	0	LS_BF_VS										
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	1	1	1	1	1	0	0	0	OUT_BUF_LOW							
W																
Reset	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 12-17. THRESHOLDS Register

Table 12-21. THRESHOLDS Field Descriptions

Field	Description
LS_BF_VS	Lines Before Vsync Threshold. It's a threshold value used to generate the LS_BF_VS interrupt status. Sets the number of lines ahead of vertical front porch (FP_V) when the interrupt is generated.
OUT_BUF_LOW	Output Buffer Filling Low Threshold (in pixels). It's used to generate the buffer under run exception. An underrun exception is generated if display needs data and output buffer filling is lower than or equal to the OUT_BUF_LOW threshold.

1. The reserved fields {bit 31, bit 15-8} should always be written with the value of 0x1f8, same value it reset to.

12.3.3.18 INT_STATUS Register

INT_STATUS register indicates the interrupt status. DIU has only one interrupt signal. The CPU reads the INT_STATUS register to decide which exception occurs when an interrupt is detected. The read operation also clears the register.

Offset 0x44 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	WB_PEND	LS_BF_VS	PARERR	UNDRUN	VSYNC_WB	VSYNC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 12-18. INT_STATUS Register

Table 12-22. INT_STATUS Field Descriptions

Field	Description
WB_PEND	Write Back Pending Interrupt: This interrupt is generated in mode 2 (the combined display and write back mode), while write back operation doesn't complete before the display frame parallel to it, if enabled. This is considered as an error and the user can select either to redo the write back frame or ignore it.
LS_BF_VS	Lines before VSYNC interrupt. This interrupt is generated threshold LS_BF_VS number of lines ahead of the vertical front porch (FP_V), if enabled.
PARERR	Display parameter error interrupt. This interrupt is generated if the user sets the display parameters wrongly, if enabled.
UNDRUN	Under run exception interrupt. This interrupt is generated when display needs data and output buffer filling is lower than or equal to the OUT_BUF_LOW threshold, if enabled.
VSYNC_WB	Vertical Synchronize Interrupt for write back operation. This interrupt is generated at the end of a write back frame, if enabled. Used in mode 2 and 3 only.
VSYNC	Vertical synchronization interrupt. This interrupt is generated at the beginning of a frame, if enabled.

12.3.3.19 INT_MASK Register

Offset 0x48 Access: User read/write

Power Architectural Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architectural Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	M_WB_PEND	M_LS_BF_VS	M_PARERR	M_UNDRUN	M_VSYNC_WB	M_VSYNC
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 12-19. INT_MASK Register

Table 12-23. INT_MASK Field Descriptions

Field	Description
M_WB_PEND	INT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 – mask the interrupt 0 – enable the interrupt
M_LS_BF_VS	
M_PARERR	
M_UNDRUN	
M_VSYN_WB	
M_VSYNC	

12.3.3.20 COLBAR Registers

The COLBAR registers are used to generate color bars in functional test mode. Eight different pixel values are taken as input data to display eight color bars on the display. After reset, they take default values, including 0xff000000 (Black), 0xff0000ff (Blue), 0xff00ffff (Cyan), 0xff00ff00 (Green), 0xffffff00 (Yellow), 0xffff0000 (Red), 0xffff00ff (Purple), and 0xffffffff (White).

12.3.3.20.1 COLBAR_1 Register¹

Offset 0x4cAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_1_R							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_1_G								COLBAR_1_B							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-20. COLBAR_1 Register (Black)

12.3.3.20.2 COLBAR_2 Register

Offset 0x50Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_2_R							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_2_G								COLBAR_2_B							
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 12-21. COLBAR_2 Register (Blue)

1. Programming the COLBAR registers at the middle of a frame affects the display immediately, so the user should reprogram them after VSYNC interrupt is detected.

12.3.3.20.3 COLBAR_3 Register

Offset 0x54 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_3_R							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_3_G								COLBAR_3_B							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 12-22. COLBAR_3 Register (Cyan)

12.3.3.20.4 COLBAR_4 Register

Offset 0x58 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_4_R							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_4_G								COLBAR_4_B							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 12-23. COLBAR_4 Register (Green)

12.3.3.20.5 COLBAR_5 Register

Offset 0x5cAccess: User read/write

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_5_R							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_5_G								COLBAR_5_B							
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 12-24. COLBAR_5 Register (Yellow)

12.3.3.20.6 COLBAR_6 Register

Offset 0x60Access: User read/write

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_6_R							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_6_G								COLBAR_6_B							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-25. COLBAR_6 Register (Red)

12.3.3.20.7 COLBAR_7 Register

Offset 0x64 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_7_R							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_7_G								COLBAR_7_B							
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 12-26. COLBAR_7 Register (Purple)

12.3.3.20.8 COLBAR_8 Register

Offset 0x68 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	COLBAR_8_R							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	COLBAR_8_G								COLBAR_8_B							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 12-27. COLBAR_8 Register (White)

12.3.3.21 FILLING Register

FILLING register is a read-only register for debug purpose. It indicates current filling status of the input and output buffers.

Offset 0x6c Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	FILLING_OBF									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FILLING_WB				FILLING_P3				FILLING_P2				FILLING_P1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 12-28. FILLING Register

Table 12-24. FILLING Field Descriptions

Field	Description
FILLING_OBF	Filling status of the output buffer (in pixels, 24 bits per pixel).
FILLING_WB	Filling status of the write back pixel buffer (number of filled buffers out of the 8 256-byte buffers).
FILLING_P3	Filling status of plane three input pixel buffer (number of filled buffers out of the 8 256-byte buffers).
FILLING_P2	Filling status of plane two input pixel buffer (number of filled buffers out of the 8 256-byte buffers).
FILLING_P1	Filling status of plane one input pixel buffer (number of filled buffers out of the 8 256-byte buffers).

12.3.3.22 PLUT Register

The PLUT register, shown in [Figure 12-29](#), determines the priority of the DIU transactions relative to other initiators on the DDR DRAM buses. PLUT register includes the 8 Priority Look Up Table components. A 4-bit priority output is selected from this look up table according to the input buffer filling status. This register must be configured so that the DIU can escalate its priority dynamically. See [Section 12.4.12](#), “Dynamic Priority Generation” for details on how to configure it.

Offset 0x70 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PRIORITY_7				PRIORITY_6				PRIORITY_5				PRIORITY_4			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PRIORITY_3				PRIORITY_2				PRIORITY_1				PRIORITY_0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 12-29. PLUT Register

Table 12-25. PLUT Field Descriptions

Field	Description
PRIORITY_7	The Priority Look Up Table components. PRIORITY_0: The highest priority; PRIORITY_7: The lowest priority.
PRIORITY_6	
PRIORITY_5	
PRIORITY_4	
PRIORITY_3	
PRIORITY_2	
PRIORITY_1	
PRIORITY_0	

12.4 Functional Description

The DIU does not have internal frame buffers. It reads the data from the main memory (DDR DRAM) at the same rate it refreshes the display.

Besides generating all the signals required to drive the display, the DIU manages real time blending of up to three planes onto the display. Alpha blending is performed between the planes. Chroma key support is also present to help relieve the host processor from all the computational and bandwidth consuming blending tasks while simultaneously allowing the users to maintain the graphics quality required by many applications.

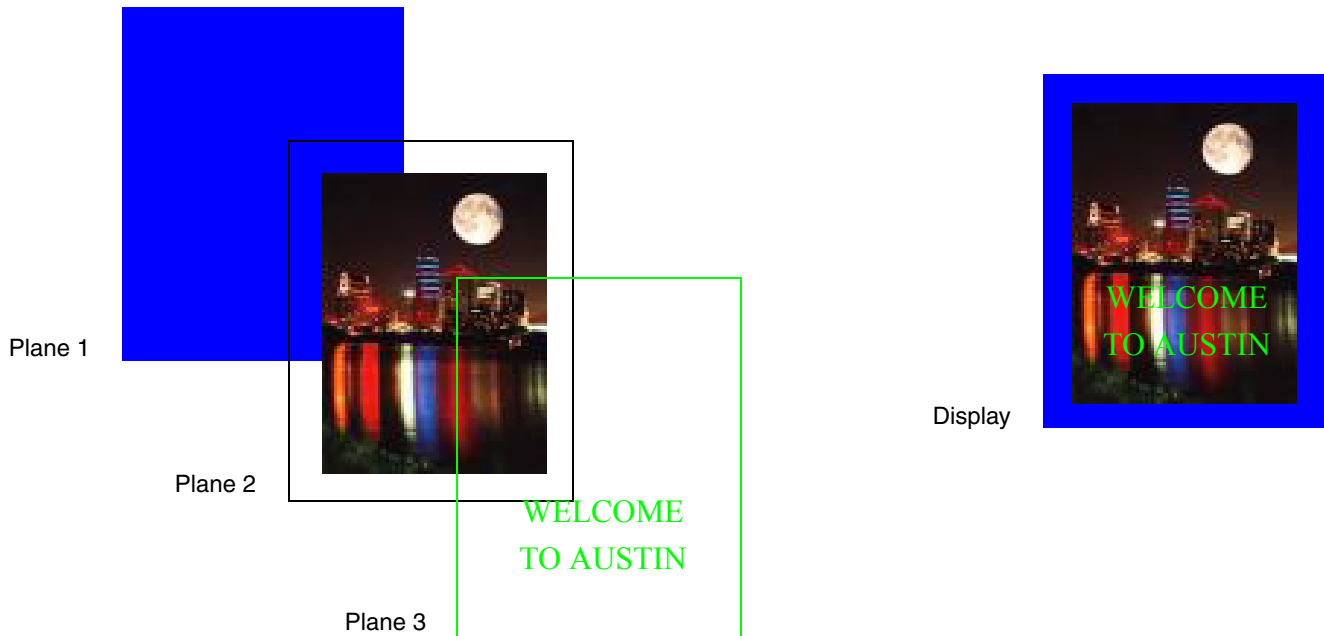


Figure 12-30. Three Plane Blending

12.4.1 Area Descriptor

The area descriptor (AD) defines each area to be displayed on a plane. A plane can display more than one area as long as they don't share a scan line.

The areas (for a plane) must be sorted in vertical order from top to bottom. The area descriptor is set up in the system's main memory (DDR DRAM) and then retrieved by the DIU directly from there.

Change the displayed data between frames by changing the data in the current area descriptor or create a new one and change the pointer in the DIU while keeping the previous one for future use or reference. It is always assumed that the bitmaps are stored pixel by pixel in memory, starting from the top-left most pixel in the image and continued sequentially until the last pixel (the bottom-right most pixel) by scanning the image always from left to right and top to bottom.

Figure 12-31 and Figure 12-32 show graphical representations of the area descriptor parameters that define an area. Figure 12-31 shows the parameters that specify how the source bitmap located in memory is interpreted by the DIU. You might want to display only a limited area of the bitmap by specifying an area of interest (AOI) as a subset of this bitmap.

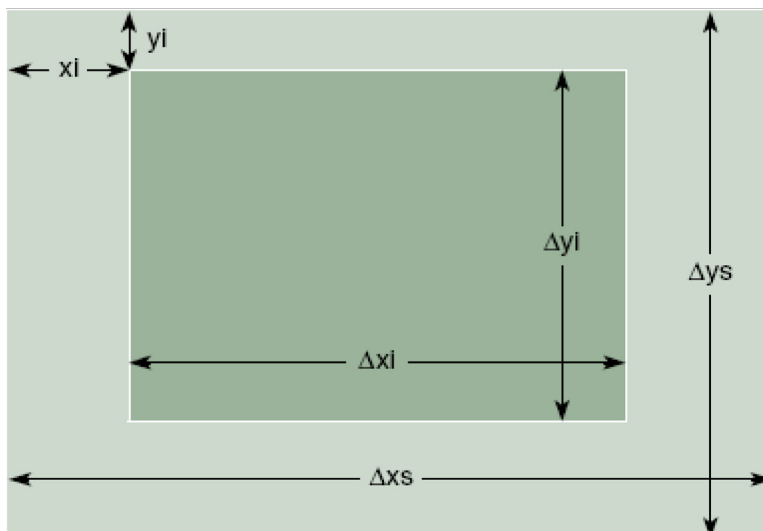


Figure 12-31. Source Bitmap Parameters

The complete source bitmap is specified by its starting address in memory, its pixel format, and its dimensions. The subset to be displayed is defined by its relative position to the beginning of the bitmap (top left corner) and its own size, which can be as large as the original image for the case where the whole image is to be displayed. The DIU automatically fetches the data, optimizing the accesses to minimize the bandwidth consumed by the operation.

There are some limitations on the size of an AOI, which include:

- The height and width of an AOI (Δy_i , Δx_i) must be greater than or equal to two pixels.
- The first AOI of a plane must be greater than or equal to 32 bytes of pixel data.

Figure 12-32 shows the parameters that specify how the image is to be displayed.

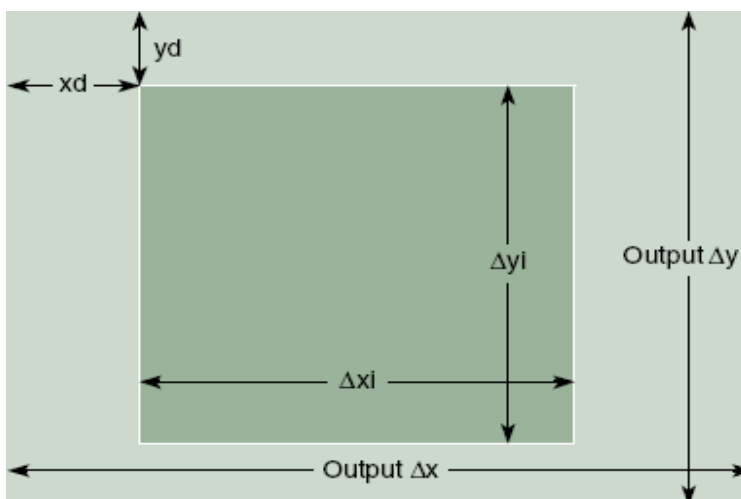


Figure 12-32. Display Parameters

There are two parameters that determine how this image is to be displayed. The first is its relative position with respect to the top-left corner of the display, and the second is its orientation (by flipping the image in its x-axis or y-axis, but not both simultaneously). The size of the display is not part of the area descriptor because it is common to all area descriptors of all planes.

12.4.2 Area Descriptor Format

Each AD is composed of a ten-word data structure. The general format for an AD is shown in [Table 12-26](#).

Table 12-26. Area Descriptor Format

Offset	General Format
0x00	Word 0 – Pixel format
0x04	Word 1 – Bitmap address
0x08	Word 2 – Source size/Global alpha
0x0C	Word 3 – AOI size
0x10	Word 4 – AOI offset
0x14	Word 5 – Display offset
0x18	Word 6 – Chroma key max
0x1C	Word 7 – Chroma key min
0x20	Word 8 – Next AD
0x24	Word 9 – Reserved

The following sections describe the individual components that make up the area descriptor.

NOTE

The area descriptor uses little-endian byte ordering. Do a 32-bit word endian swap for each word before writing it to memory.

12.4.2.1 Area Descriptor Word 0 – Pixel Format

Figure 12-33 shows the fields of AD Word 0, which defines the pixel format. Table 12-27 describes the pixel format fields.

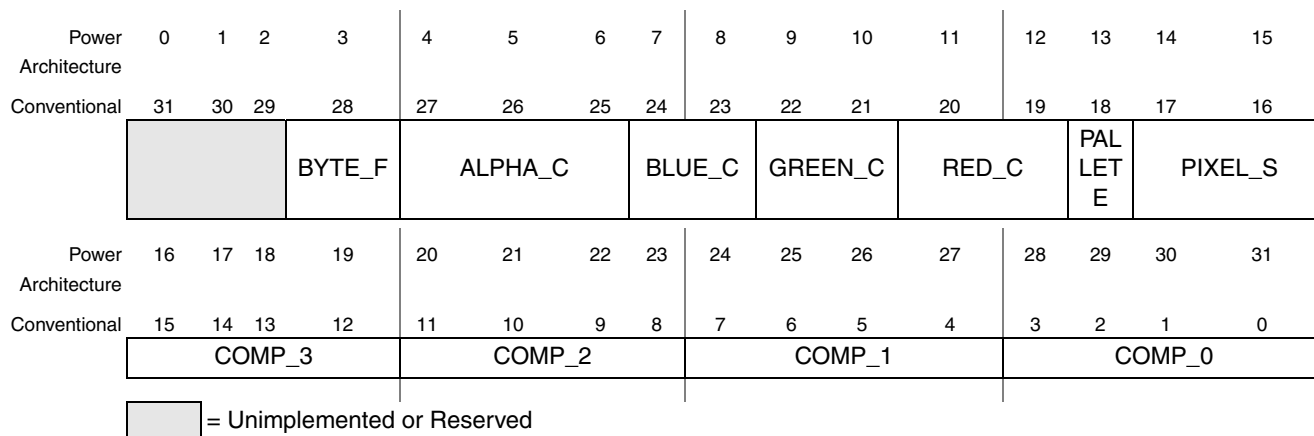


Figure 12-33. Area Descriptor Word 0 – Pixel Format

Table 12-27. Area Descriptor Word 0 – Pixel Format

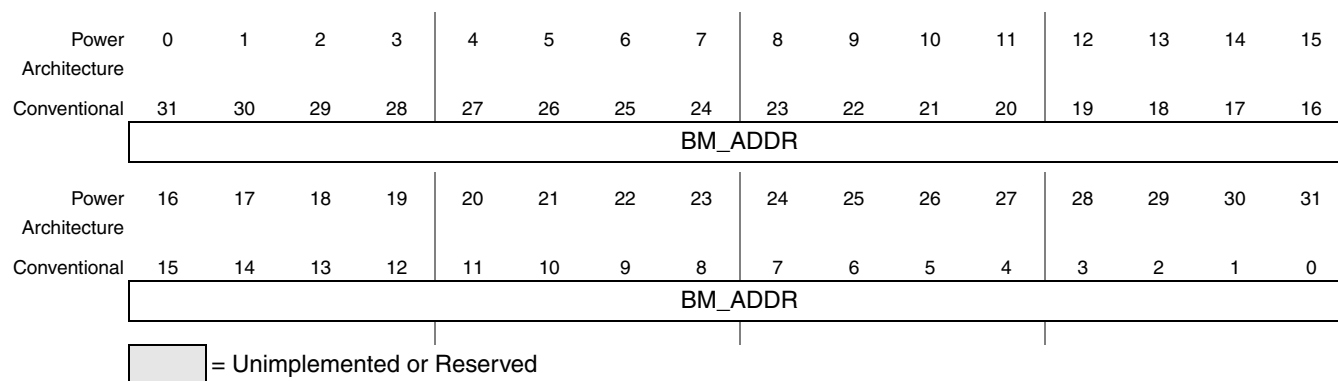
Field	Description
COMP_0	Number of bits for component 0. Valid range is 0 through 8.
COMP_1	Number of bits for component 1. Valid range is 0 through 8.
COMP_2	Number of bits for component 2. Valid range is 0 through 8.
COMP_3	Number of bits for component 3. Valid range is 0 through 8.
PIXEL_S	Pixel size. Specifies the number of bytes per pixel. The actual number of bytes per pixel is PIXEL_S+1. 00 1 byte per pixel 01 2 bytes per pixel 10 3 bytes per pixel 11 4 bytes per pixel
PALETTE	Palette mode. Determines whether palette mode is used. 0 Palette mode disabled 1 Palette mode enabled
RED_C	Red Component assignment.. This field assigns the component number used for the red channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 10.
GREEN_C	Green component assignment. This field assigns the component number used for the green channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 01.

Table 12-27. Area Descriptor Word 0 – Pixel Format (continued)

BLUE_C	Blue component assignment. This field assigns the component number used for the blue channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 00.
ALPHA_C	Alpha component assignment. This field assigns the component number used for the alpha channel. 000 Component 0 001 Component 1 010 Component 2 011 Component 3 100 Global alpha Default value is 011.
BYTE_F	Byte flip disable. When cleared, flips the byte order for the pixel data. See Section 12.4.3, “Pixel Structure for more information. 0 Bytes are flipped 1 Bytes are not flipped

12.4.2.2 Area Descriptor Word 1 – Bitmap Address

[Figure 12-34](#) shows the fields of AD Word 1, which defines the bitmap address. [Table 12-28](#) describes the bitmap address fields.

**Figure 12-34. Area Descriptor Word 1 – Bitmap Address****Table 12-28. Area Descriptor Word 1 – Bitmap Address**

Field	Description
BM_ADDR	Bitmap address pointer. It points to the bitmap data in memory.

12.4.2.3 Area Descriptor Word 2 – Source Size/Global Alpha

Figure 12-35 shows the fields of AD Word 2, which defines the source size and the global alpha value. Table 12-29 describes the source size/global alpha fields.

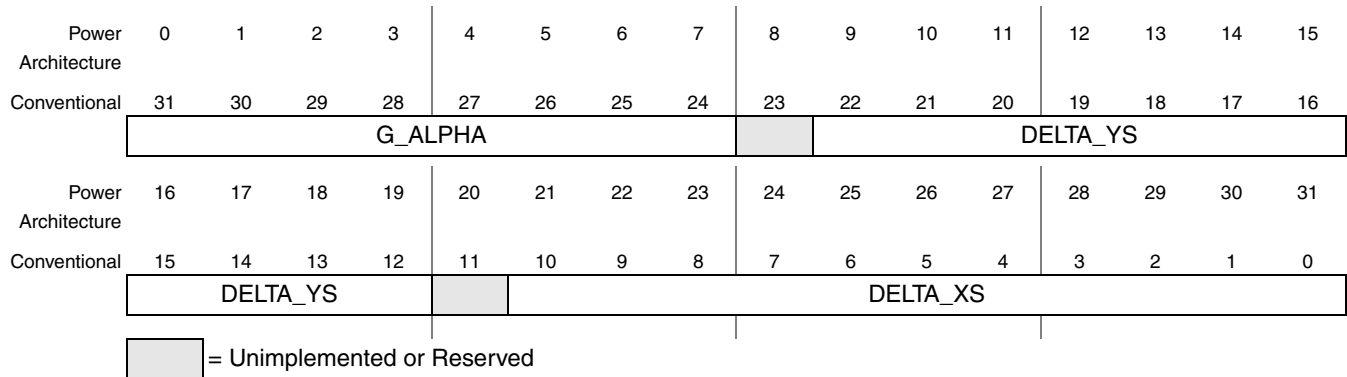


Figure 12-35. Area Descriptor Word 2 – Source Size/Global Alpha

Table 12-29. Area Descriptor Word 2 – Source Size/Global Alpha

Field	Description
DELTA_XS	The Δx_s parameter that defines the horizontal size (in pixels) of the source bitmap.
DELTA_YS	The Δy_s parameter that defines the vertical size (in pixels) of the source bitmap.
G_ALPHA	Global alpha. Value used for the alpha channel for all pixels in the area when the data format does not include an alpha component or when the user wants to override the alpha values in the data (ALPHA_C = 100). Pixels to be displayed on Plane 1 ignore the alpha value.

12.4.2.4 Area Descriptor Word 3 – AOI Size

Figure 12-36 shows the fields of AD Word 3, which defines the size of the area of interest. Table 12-30 describes the area of interest size fields.

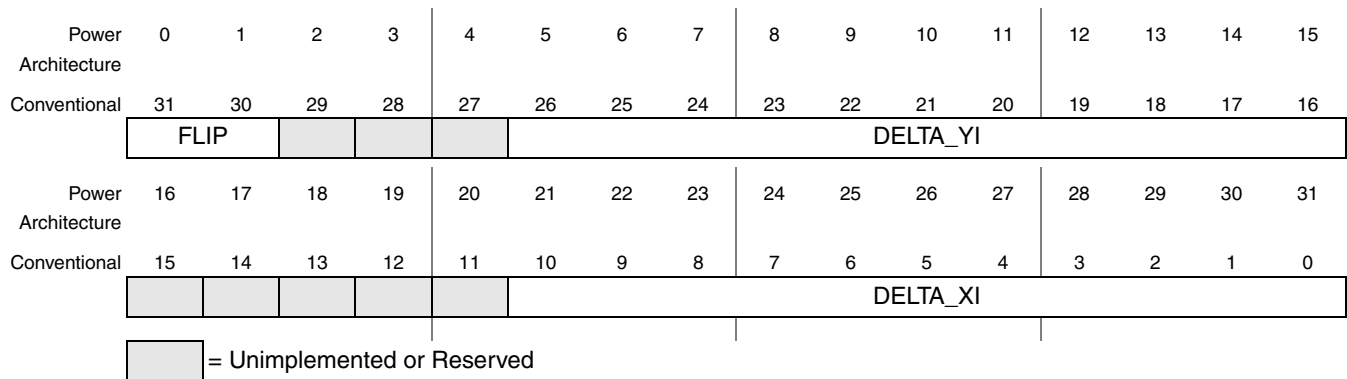


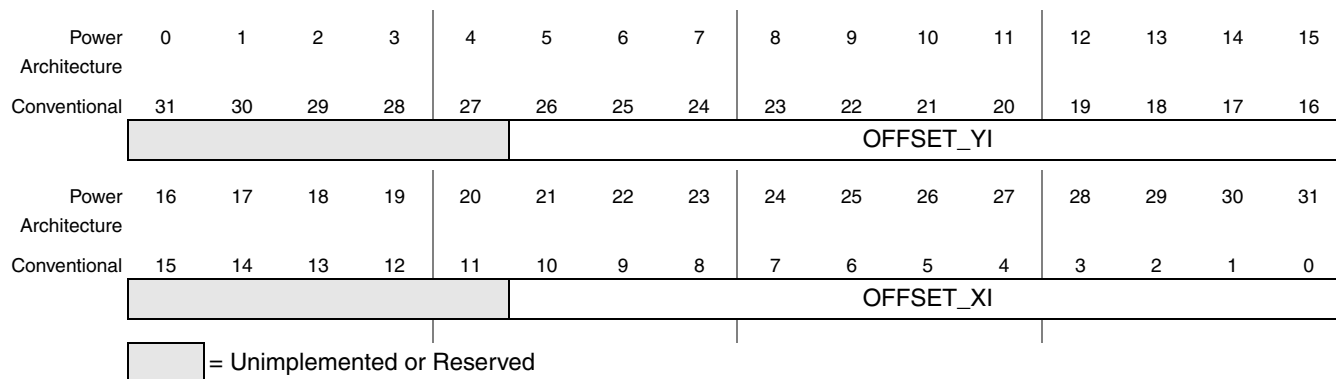
Figure 12-36. Area Descriptor Word 3 – AOI Size

Table 12-30. Area Descriptor Word 3 – AOI Size

Field	Description
DELTA_XI	The Δxi parameter that defines the horizontal size (in pixels) of the area of interest.
DELTA_XI	The Δxi parameter that defines the horizontal size (in pixels) of the area of interest.
FLIP	Area of interest image flip. Flips the bitmap image either horizontally or vertically within the area of interest. 00 Normal 01 Flip image horizontally (about the y-axis) 10 Flip image vertically (about the x-axis) 11 Reserved

12.4.2.5 Area Descriptor Word 4 – AOI Offset

Figure 12-37 shows the fields of AD Word 4, which defines the offset for the area of interest. Table 12-31 describes the area of interest offset fields.

**Figure 12-37. Area Descriptor Word 4 – AOI Offset****Table 12-31. Area Descriptor Word 4 – AOI Offset**

Field	Description
OFFSET_XI	The xi parameter that defines the horizontal offset (in pixels) of the area of interest from the start of the source bitmap in memory.
OFFSET_XI	The xi parameter that defines the horizontal offset (in pixels) of the area of interest from the start of the source bitmap in memory.

12.4.2.6 Area Descriptor Word 5 – Display Offset

Figure 12-38 shows the fields of AD Word 5, which defines the offset for the area of interest in the display. Table 12-32 describes the display offset fields.

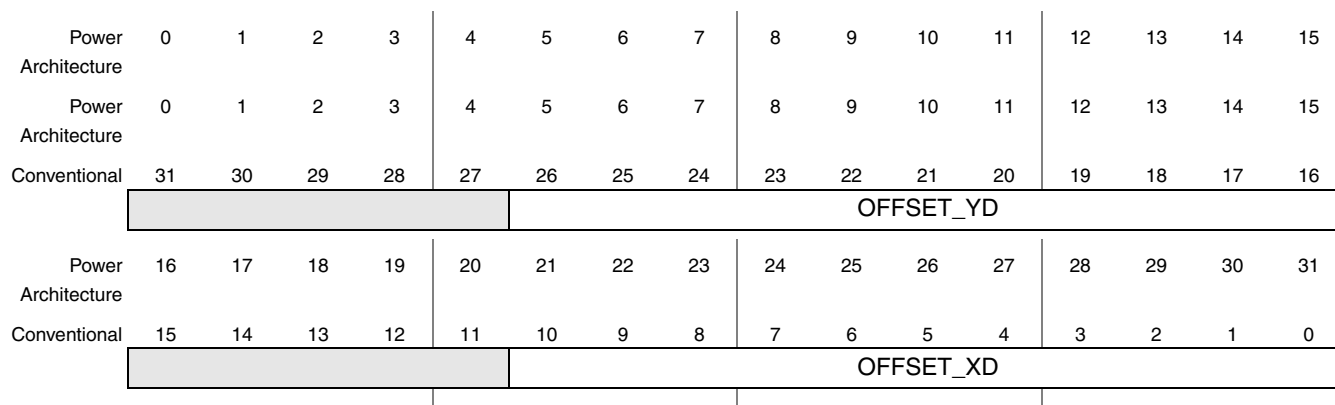


Figure 12-38. Area Descriptor Word 5 – Display Offset

Table 12-32. Area Descriptor Word 5 – Display Offset

Field	Description
OFFSET_XD	The xd parameter that defines the horizontal offset (in pixels) of the area of interest in the display.
OFFSET_YD	The yd parameter that defines the vertical offset (in pixels) of the area of interest in the display.

12.4.2.7 Area Descriptor Word 6 – Chroma Key Max

Figure 12-39 shows the fields of AD Word 6, which defines the maximum values for chroma key. See Section 12.4.6, “Chroma Keying” for more information. Table 12-33 describes the chroma key max fields.

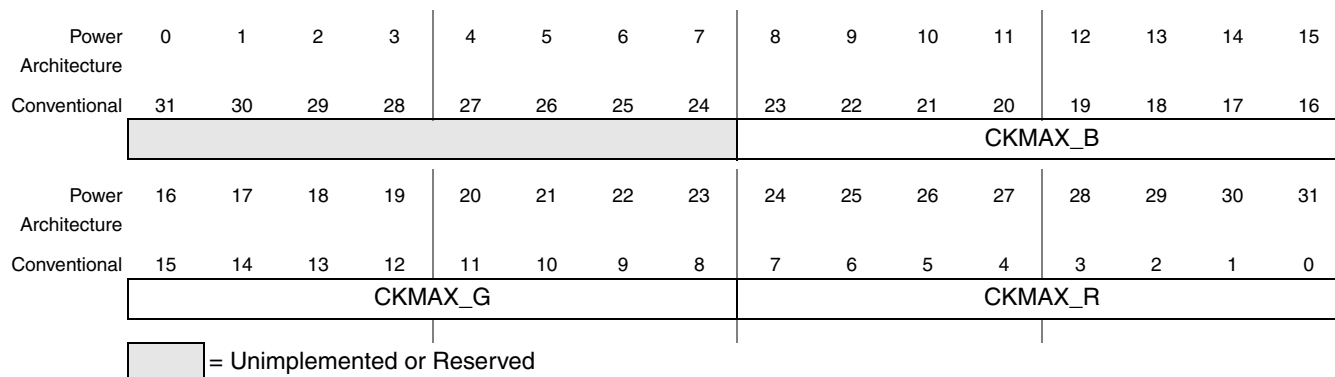


Figure 12-39. Area Descriptor Word 6 – Chroma Key Max

Table 12-33. Area Descriptor Word 6 – Chroma Key Max

Field	Description
CKMAX_R	Chroma key maximum red component value
CKMAX_G	Chroma key maximum green component value
CKMAX_B	Chroma key maximum blue component value

12.4.2.8 Area Descriptor Word 7 – Chroma Key Min

Figure 12-40 shows the fields of AD Word 7, which defines the minimum values for chroma key. See Section 12.4.6, “Chroma Keying” for more information. Table 12-34 describes the chroma key min fields.

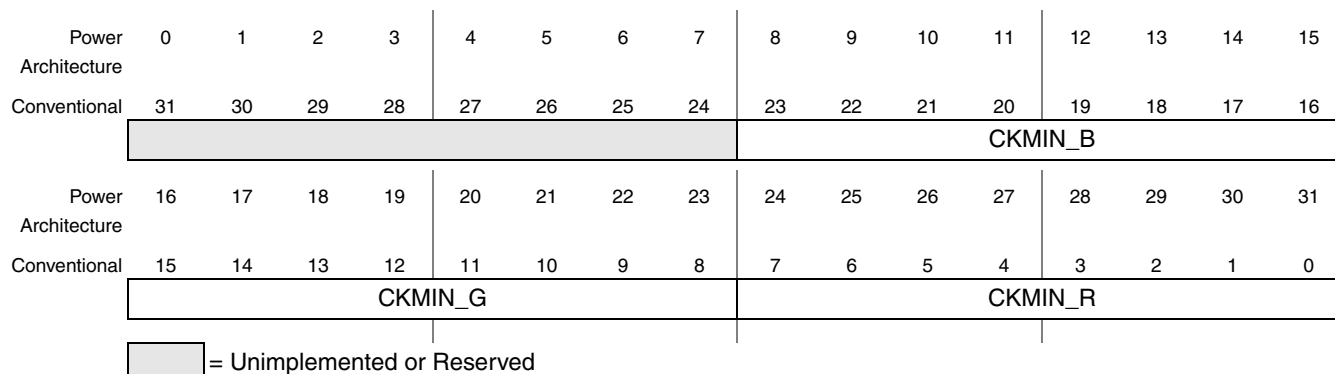


Figure 12-40. Area Descriptor Word 7 – Chroma Key Min

Table 12-34. Area Descriptor Word 7 – Chroma Key Min

Field	Description
CKMIN_R	Chroma key minimum red component value
CKMIN_G	Chroma key minimum green component value
CKMIN_B	Chroma key minimum blue component value

12.4.2.9 Area Descriptor Word 8 – Next AD

Figure 12-41 shows the fields of AD Word 8, which defines the next AD address. If more than one area is to be displayed in the same plane, the next AD points to the next area descriptor. If this is the last AD in the current frame, the next AD address must be set to 0x0000_0000. The next AD address must be aligned to an 8-byte boundary (that is, the lowest three bits should be 000). Table 12-35 describes the next area descriptor address fields.

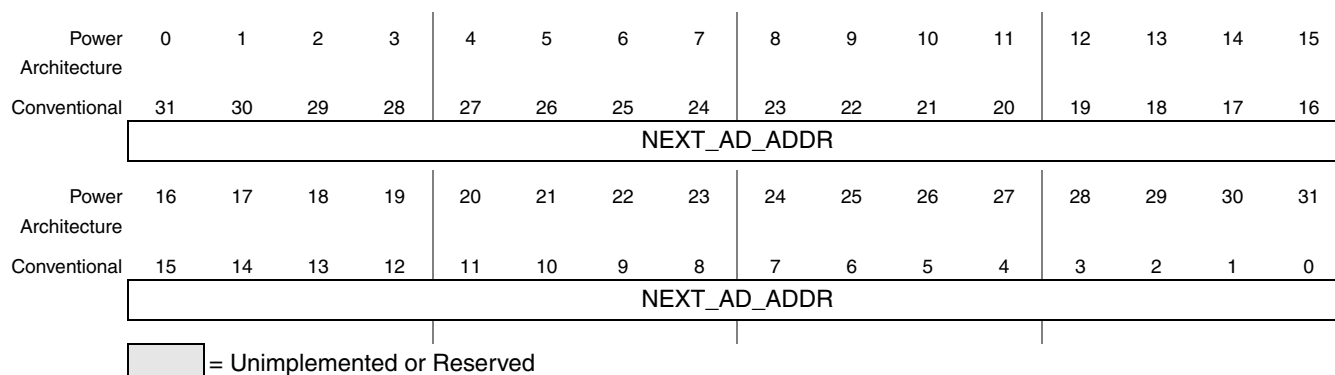


Figure 12-41. Area Descriptor Word 8 – Next AD

Table 12-35. Area Descriptor Word 8 – Next AD

Field	Description
-------	-------------

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NEXT_AD_ADDR																
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEXT_AD_ADDR																
	= Unimplemented or Reserved															
NEXT_AD_ADDR	The next Area Descriptor address pointer. It's used to point to the next AD in memory when more than one area is to be displayed in a plane. This field must be set to 0x0000_0000 if this is the last AD in the current frame. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0).															

12.4.3 Pixel Structure

The DIU has been designed for maximum flexibility in the format of its input data. Each pixel can contain up to four basic components: alpha, red, green, and blue. The contents of several registers in an area descriptor (see [Figure 12-33](#)) determine how the DIU parses the bitmap data into the pixel components.

The first step is to define the size of each pixel (PIXEL_S+1) and the number of bits for each component of a pixel (COMP_0 to COMP_3) in the data stream. The next step is to assign the components of the pixel (alpha, R, G, and B) to the components of the data stream (COMP_0 to COMP_3). The RED_C, GREEN_C, BLUE_C, and ALPHA_C fields in the pixel format (word 0) of the AD are used to control the assignments, selecting which component is red, which is green and so on. The three color components (red, green, and blue) must be mapped to one of the components of the data stream; alpha can be assigned to a fourth component of the data stream or the pixel can use the global alpha value specified for the current area descriptor (G_ALPHA in word 2).

The pixel format (word 0) of the AD also contains the BYTE_F bit, which can have a significant impact on the way the DIU interprets the pixel data. If the BYTE_F bit in the AD is set (that is, flipping is disabled) then the pixel is constructed as shown in [Figure 12-42](#).

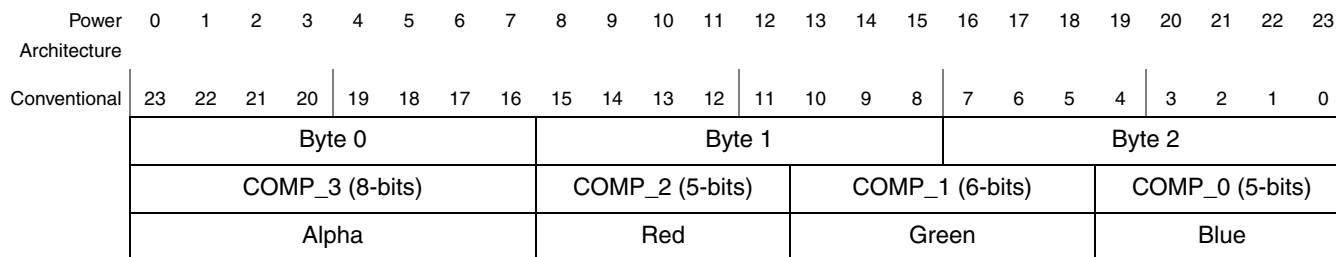


Figure 12-42. 24-bit ARGB 8:5:6:5 Pixel Structure Definition when BYTE_F = 1

If the BYTE_F bit in the AD is cleared (that is, bytes in the pixel are flipped) then the pixel is constructed as shown in [Figure 12-43](#). The DIU performs the byte flipping on each pixel before it performs the mapping.

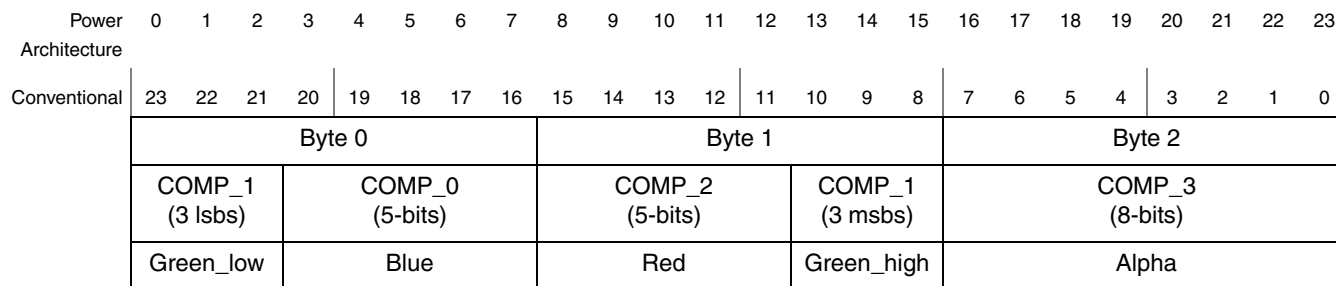


Figure 12-43. 24-bit ARGB 8:5:6:5 Pixel Structure Definition when BYTE_F = 0

12.4.4 Pixel Format Conversion

The DIU is designed to support a variety of pixel bit formats. [Table 12-36](#) lists some examples of DIU supported pixel formats.

Table 12-36. Examples of DIU Supported Pixel Formats

Component Order 3, 2, 1, 0	Number of Bits per Component	G.Alpha	Palette
A:R:G:B	8:8:8:8	No	No
R:G:B	8:8:8	Yes	No
A:R:G:B	8:5:6:5	No	No
R:G:B	5:6:5	Yes	No
A:R:G:B	4:4:4:4	No	No
A:R:G:B	1:5:5:5	No	No
n/a	n/a	no	Yes
R:G:B:A	8:8:8:8	No	No
R:G:B:A	5:6:5:8	No	No
R:G:B:A	4:4:4:4	No	No
R:G:B:A	5:5:5:1	No	No
n/a	8	Yes	No

Internally, all the calculations are performed using pixels represented by 8 bits per component. For those pixel formats you specify less than 8 bits per color components, the specified bits are used as MSB and the LSB are filled with zeros. The alpha values are extended to 8 bits too, but the LSB are filled by sign extension to make sure that minimum value is 0 and the maximum value is extended to 255.

If the original data bitmap does not include alpha, you can specify it in the global alpha (G_ALPHA) field of the area descriptor and/or use chroma keying.

12.4.4.1 Palette Mode

As an alternate to the larger bitmap formats, the DIU supports an 8-bit pixel format through a palette table (256 x 32-bit look up table). The palette table maps an 8-bit input pixel to a 32-bit color format. The palette table is stored in a 256 x 32 bit embedded SRAM. It is accessed by the hardware as an indexed matrix such that `palettized_pixel [31-0] = palette_table[input_pixel [7-0]]`.

The palette table should be created in advance in memory (DDR DRAM) as a consecutive sequence of 256 x 64 bits (with bits 32 to 63 filled with zeros). Writing to the PALETTE register ([12.3.3.5/12-12](#)) causes the DIU to automatically reload a new palette table from memory before it starts processing the next frame. [Figure 12-44](#) shows the format of the palette table in main memory.

Palette Entry	PALETTE Offset	Local Address [2:0]							
		000	001	010	011	100	101	110	111
0	0x000	P0_B	P0_G	P0_R	P0_A	0x00	0x00	0x00	0x00
1	0x008	P1_B	P1_G	P1_R	P1_A	0x00	0x00	0x00	0x00
2	0x010	P2_B	P2_G	P2_R	P2_A	0x00	0x00	0x00	0x00
3	0x018	P3_B	P3_G	P3_R	P3_A	0x00	0x00	0x00	0x00
.....									
254	0x7F0	P254_B	P254_G	P254_R	P254_A	0x00	0x00	0x00	0x00
255	0x7F8	P255_B	P255_G	P255_R	P255_A	0x00	0x00	0x00	0x00

Figure 12-44. Palette Table Format in Memory

12.4.5 Alpha Blending

For each pixel, besides the color components (R, G, B), there is a fourth component that defines the transparency of the pixel. This transparency value is called alpha and has a range between 0 (pixel is completely transparent) to 255 (pixel is completely opaque). The following equation represents the transfer function of the blending operation including the alpha values. There is no alpha component defined for plane1 because there are no planes behind it.

$$\text{out_pixel} = \frac{\text{pixel_1} \times (255 - \alpha_2) \times (255 - \alpha_3) + \text{pixel_2} \times \alpha_2 \times (255 - \alpha_3) + \text{pixel_3} \times 255 \times \alpha_3}{255^2} \quad \text{Eqn. 12-1}$$

For mode 2, when only plane 2 and 3 are blended to write back, the equation changes to:

$$\text{writeback} = \frac{\text{plane2} \times (255 - \alpha_3) + \text{plane3} \times \alpha_3}{255} \quad \text{Eqn. 12-2}$$

12.4.6 Chroma Keying

For each area (see [Section 12.4.1, “Area Descriptor”](#)), specify a maximum and a minimum value for the chroma keying function. The maximum and minimum values are specified as R,G, and B values to which every pixel in the bitmap is compared and if all the components are greater or equal to the minimum and less than or equal to the max then the alpha component for that particular pixel is replaced with 0. The chroma keying operation is performed after the color format conversion with all the components extended to 8 bits.

To turn the chroma keying off, set the minimum to 255 and the max to 0, for each component. To produce a green-screen type of effect in which all green pixels (0,255,0) are to be turned transparent, the max and min should both be set to (0,255,0).

12.4.7 Gamma Correction

The gamma table allows the user to define a completely arbitrary transfer function at the output of each color component. The gamma table should be created in memory as a consecutive sequence of 256 bytes for each color component accessed by the hardware as an indexed matrix so that $\text{output_color_component} = \text{gamma_table}[\text{input_color_component}]$.

All three gamma tables must be stored in memory (DDR DRAM) consecutively beginning with red, then green, and blue at the end.

Similar to the palette, the gamma table is also automatically loaded by the DIU from memory into an embedded SRAM before it starts on processing the next frame if the CPU writes to the GAMMA register (12.3.3.4/12-11).

The size of the SRAM is 3 x 256 bytes, but it's organized as 96 x 64 bits. So address range for each color component table is,:

- Gamma_red: 0x0~0xff,
- Gamma_green: 0x100~0x1ff
- Gamma_blue: 0x200~0x2ff.

Figure 12-45 shows the format of the gamma table in main memory.

Color Component	GAMMA Offset	Local Address [2:0]							
		000	001	010	011	100	101	110	111
Red	0x000	G0 _{red}	G1 _{red}	G2 _{red}	G3 _{red}	G4 _{red}	G5 _{red}	G6 _{red}	G7 _{red}
	0x008	G8 _{red}	G9 _{red}	G10 _{red}	G11 _{red}	G12 _{red}	G13 _{red}	G14 _{red}	G15 _{red}
								
	0x0F0	G240 _{red}	G241 _{red}	G242 _{red}	G243 _{red}	G244 _{red}	G245 _{red}	G246 _{red}	G247 _{red}
	0x0F8	G248 _{red}	G249 _{red}	G250 _{red}	G251 _{red}	G252 _{red}	G253 _{red}	G254 _{red}	G255 _{red}
Green	0x100	G0 _{green}	G1 _{green}	G2 _{green}	G3 _{green}	G4 _{green}	G5 _{green}	G6 _{green}	G7 _{green}
	0x108	G8 _{green}	G9 _{green}	G10 _{green}	G11 _{green}	G12 _{green}	G13 _{green}	G14 _{green}	G15 _{green}
								
	0x1F0	G240 _{green}	G241 _{green}	G242 _{green}	G243 _{green}	G244 _{green}	G245 _{green}	G246 _{green}	G247 _{green}
	0x1F8	G248 _{green}	G249 _{green}	G250 _{green}	G251 _{green}	G252 _{green}	G253 _{green}	G254 _{green}	G255 _{green}
Blue	0x200	G0 _{blue}	G1 _{blue}	G2 _{blue}	G3 _{blue}	G4 _{blue}	G5 _{blue}	G6 _{blue}	G7 _{blue}
	0x208	G8 _{blue}	G9 _{blue}	G10 _{blue}	G11 _{blue}	G12 _{blue}	G13 _{blue}	G14 _{blue}	G15 _{blue}
								
	0x2F0	G240 _{blue}	G241 _{blue}	G242 _{blue}	G243 _{blue}	G244 _{blue}	G245 _{blue}	G246 _{blue}	G247 _{blue}
	0x2F8	G248 _{blue}	G249 _{blue}	G250 _{blue}	G251 _{blue}	G252 _{blue}	G253 _{blue}	G254 _{blue}	G255 _{blue}

Figure 12-45. Gamma Table Format in Memory

12.4.8 Cursor

The DIU supports a 32 x 32 hardware cursor that is overlapped on top of all three planes. [Figure 12-46](#) shows the structure of the cursor bitmap in memory (DDR DRAM). In [Figure 12-47](#), each cursor pixel is labeled as $C(n,m)$ where n indicates the row and m indicates the column of the 32 x 32 matrix. The cursor bitmap data is stored in memory as a continuous sequence of 16-bit pixels, starting from the top left corner, $C(0,0)$, and continuing across the row (left to right) and then on to the next column (top to bottom) until the last cursor pixel, $C(31,31)$, is at the bottom right corner.

Cursor Pixel Row	CURSOR Offset	Local Address [2:0]							
		000	001	010	011	100	101	110	111
0	0x000	C(0,0)		C(0,1)		C(0,2)		C(0,3)	
	0x008	C(0,4)		C(0,5)		C(0,6)		C(0,7)	
								
	0x038	C(0,28)		C(0,29)		C(0,30)		C(0,31)	
1	0x040	C(1,0)		C(1,1)		C(1,2)		C(1,3)	
	0x048	C(1,4)		C(1,5)		C(1,6)		C(1,7)	
								
	0x078	C(1,28)		C(1,29)		C(1,30)		C(1,31)	
2	0x080	C(2,0)		C(2,1)		C(2,2)		C(2,3)	
	0x088	C(2,4)		C(2,5)		C(2,6)		C(2,7)	
								
	0x0B8	C(2,28)		C(2,29)		C(2,30)		C(2,31)	
.....									
31	0x7C0	C(31,0)		C(31,1)		C(31,2)		C(31,3)	
								
	0x7F0	C(31,24)		C(31,25)		C(31,26)		C(31,27)	
	0x7F8	C(31,28)		C(31,29)		C(31,30)		C(31,31)	

Figure 12-46. Cursor Structure in Memory

Each cursor pixel is in 16-bit, GLBARGH 3:5:1:5:2 format, as shown in [Figure 12-47](#). The field descriptions for a cursor pixel are provided in [Table 12-37](#).

NOTE

The cursor pixel uses little-endian byte ordering. Do a 16-bit halfword endian swap for each pixel while rendering the cursor bitmap in software.

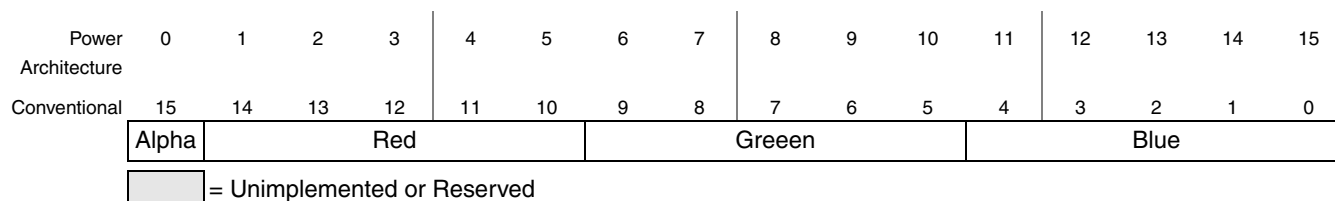


Figure 12-47. Cursor Pixel Format

Table 12-37. Field Descriptions for Cursor Pixel

Field	Description
Alpha	Alpha component of cursor pixel $C(n,m)$.
Red	Red component of cursor pixel $C(n,m)$.
Green	Green component of the cursor pixel is made up of $Green_{High} Green_{Low}$.
Blue	Blue component of cursor pixel $C(n,m)$.

To achieve cursors with shapes other than the basic square set the alpha value to 0 in those pixels that are to be transparent. To make the cursor disappear set the alpha value to 0 for all the pixels.

Similar to the palette and gamma, the cursor bitmap is also automatically loaded by the DIU from memory into an embedded SRAM before it starts on processing the next frame if the CPU write to the CURSOR register (See [Section 12.3.3.6, “CURSOR Register](#)). The SRAM size is, 256 x 64 bits.

The cursor position is determined by the register CURSOR_POS (See [Section 12.3.3.7, “CURS_POS Register](#)); the specified coordinate corresponds to the top, left corner of the cursor bitmap. The cursor can be partially or totally off the screen.

12.4.9 Write Back Operation

Besides driving the LCD display, the DIU supports two operating modes (mode 2 and 3) with memory write-back operation. Intermediate blending results are stored back into the memory (DDR DRAM). These intermediate results can be then forwarded to other processing or display units or blended with other image source(s) again in the DIU, virtually extending the number of graphics planes.

DIU mode 1 supports the blending and display of up to 3 planes. If an application requires the display of an image blended from four or more planes, this can be accomplished in DIU mode 2. Plane 1 is used to display one frame while the following frame is blended by the DIU. Planes 2 and 3 are used initially to read in the first and second planes to be combined, and the DIU writes back a combined plane 1+2. Then, while the DIU displays the same frame on plane 1, the DIU reads back in on plane 2 the intermediate plane 1+2, and on plane 3 the new third plane, writing back a combined plane 1+2+3. Then (while the DIU

displays the same frame on plane 1), the DIU reads back in on plane 2 the intermediate plane 1+2+3, and on plane 3 the new fourth plane, writing back the combined plane 1+2+3+4. This combined plane is then ready to be displayed on plane 1 in the following frame, or extra steps can be added to blend more than four planes.

The write back pixel format is always 24-bit RGB 8:8:8. stored in the memory continuously and packed. [Figure 12-48](#) shows the format of the write-back pixels in main memory.

Memory Offset	Local Address [2:0]							
	000	001	010	011	100	101	110	111
0x00	B0	G0	R0	B1	G1	R1	B2	G2
0x08	R2	B3	G3	R3	B4	G4	R4	B5
0x10	G5	R5	B6	G6	R6	B7	G7	R7
0x18	B8	G8	R8	B9	G9	R9	B10	G10
0x20	R10

Figure 12-48. Write-Back Pixel Format in Memory

The register WB_MEM_ADDR ([12.3.3.13/12-19](#)) specifies the address in the memory where the write back data should be written to. Writing to this address triggers a write back frame refresh. A VSYNC_WB interrupt occurs at the end of a write back frame if enabled.

Write back frame size is specified by the WB_SIZE register ([12.3.3.12/12-18](#)).

NOTE

In mode 2, the DIU works on the basis that the writeback frame completes before the end of the display frame parallel to it (i.e. before the vertical front porch). A WB_PEND interrupt can be generated if enabled, indicating the writeback operation doesn't complete in time.

12.4.10 Color Bar Generation

For testing purposes, it is desirable to generate color bars within the DIU itself. This is achieved by setting the DIU_MODE register to 4. The pattern produced is a simple one with eight vertical color bars. This mode allows the user to verify that the DIU is operational without the need to interact with the system memory. A basic color sequence is preloaded, but the user can overwrite the default values if needed. See [Section 12.3.3.20, “COLBAR Registers](#) for more information.

The size of the bars is set by dividing the horizontal resolution by 8 using integer math. If the horizontal resolution is not divisible by 8 exactly, not all color bars have the same width.

12.4.11 Interrupt Generation

The DIU generates interrupt through a single line controlled by the contents of two registers: INT_STATUS ([12.3.3.18/12-24](#)) and INT_MASK ([12.3.3.19/12-25](#)). When an interrupt occurs, the host

needs to read the INT_STATUS register to find the source of the interrupt. The read operation also clears the register.

There are six defined interrupt statuses: VSYNC -- Vertical Synchronization, VSYNC_WB -- Vertical Synchronization for Write Back, UNDRUN -- Under Run Exception, PARERR -- Display Parameter Error, LS_BF_VS -- Lines Before VSYNC, and WB_PEND -- Writeback Pending.

A display parameter error interrupt is generated if the user set the display parameters incorrectly.

Table 12-38 gives a list of parameter error conditions under different DIU operating mode. When a PARERR interrupt is detected, the user can select turn off the DIU (program the DIU_MODE to 0), correct the wrong parameter(s), and turn it on again. The DIU is initialized internally.

Table 12-38. Parameter Error Conditions

Mode 1 & 2	$\text{DELTA_XI} > \text{DELTA_XS}$
	$\text{DELTA_YI} > \text{DELTA_YS}$
	$\text{DELTA_XI} + \text{OFFSET_XD} > \text{DELTA_X}$
	$\text{DELTA_YI} + \text{OFFSET_YD} > \text{DELTA_Y}$
	$\text{DELTA_XI} + \text{OFFSET_XI} > \text{DELTA_XS}$
	$\text{DELTA_YI} + \text{OFFSET_YI} > \text{DELTA_YS}$
Mode 2 & 3	$\text{DELTA_XI} > \text{DELTA_X_WB}$
	$\text{DELTA_YI} > \text{DELTA_Y_WB}$
	$\text{DELTA_XI} + \text{OFFSET_XD} > \text{DELTA_X_WB}$
	$\text{DELTA_YI} + \text{OFFSET_YD} > \text{DELTA_Y_WB}$
	$\text{DELTA_XI} + \text{OFFSET_XI} > \text{DELTA_XS}$
	$\text{DELTA_YI} + \text{OFFSET_YI} > \text{DELTA_YS}$

12.4.12 Dynamic Priority Generation

The multi-port DRAM controller has built-in arbiters that use a 4-bit priority signal (i.e. 16 levels of priority). The DRAM controller tries to service the request with the highest priority first. The task of setting the priorities is done by the DRAM Controller Priority Manager block, the priority manager. It dynamically sets the priorities of the DRAM busses in such a way that bandwidth is divided fairly and each master receives its fair share of the bandwidth. Programming the look-up tables in the priority manager allows controlling relative priority to other channels and the average share of the bandwidth the current master gets.

The DIU outputs a 4-bit priority signal to the priority manager. The priority is a function of the internal input buffer filling level (buffer full → low priority, buffer empty → high priority). The buffer filling level ranges from 0 to 7 and it's used to select a LUT component from the PLUT register as the priority. Filling level 0 selects PRIORITY_0 and filling level 7 selects PRIORITY_7.

The priority manager can be programmed to take the DIU priority output directly (option 1) or to follow the normal priority schema (option 2). It's recommended to use option 2, the default option.

To use option 1, the user should set `PRIOMAN_CONFIG2[DIU-OVERRULE]` (refer to DRAM Controller Priority Manager chapter), and program a set of priority values in the PLUT register (`PRIORITY_0` to `PRIORITY_7` from high to low). The priority value ranges from 0 to 15 and accesses are blocked if the priority value is 0.

To use option 2, it's recommended to program `PRIOMAN_CONFIG2[LUT SEL0]` to 3, so that the priority manager selects the alternate look-up table for the DIU if DIU incoming priority bit 3 is high. So the user can set the DIU priority low in the main look-up table and set it high in the alternate look-up table, and program the PLUT register so for example its priority bit 3 is high (0x8) when the buffer is lower than half full. The purpose to do this is to only escalate the DIU priority when necessary, and save the DRAM bandwidth to the other masters like the Power Architecture e300 core.

12.4.13 Display Signal Timing

The first step to generate appropriate timing signals for the selected display is to adjust the frequency of the pixel clock to a frequency within the specified parameters of the display (see [Section 12.4.13.1, “Refresh Rate”](#)). Program the `SCFR1` register in the system clock module, `SCFR1[DIU_DIV]`, to set the divide ratio for the DIU pixel clock (refer to the system clock chapter).

Then the horizontal and vertical synchronize signals are generated based on the pixel clock. The relationship between pixel clock, HSYNC, and VSYNC signals is shown in diagram [Figure 12-49](#) and [Figure 12-50](#).

Refer to [Section 12.3.3.11, “DISP_SIZE Register”](#), [Section 12.3.3.14, “HSYN_PARA Register”](#), [Section 12.3.3.15, “VSYN_PARA Register”](#), [Section 12.3.3.16, “SYN_POL Register”](#) for related display parameter configuration registers.

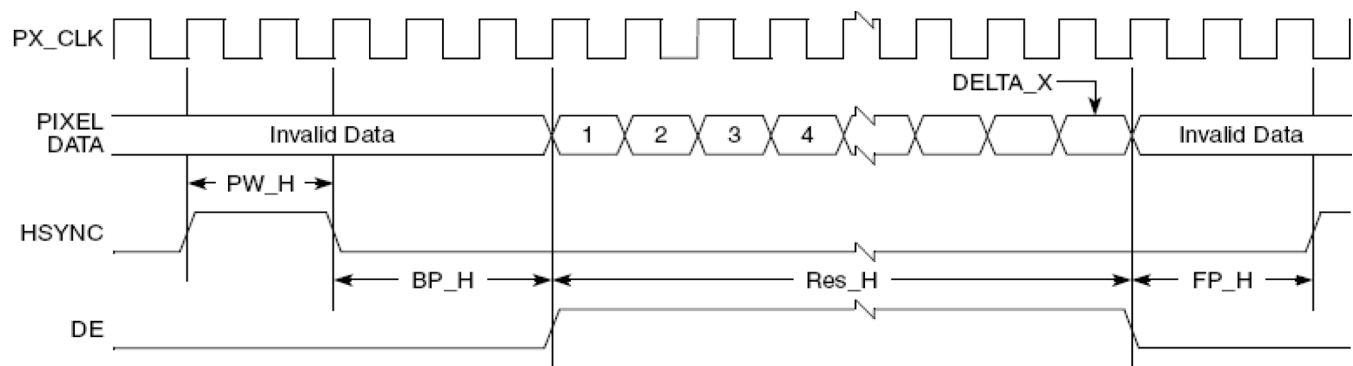


Figure 12-49. Horizontal Sync Signals

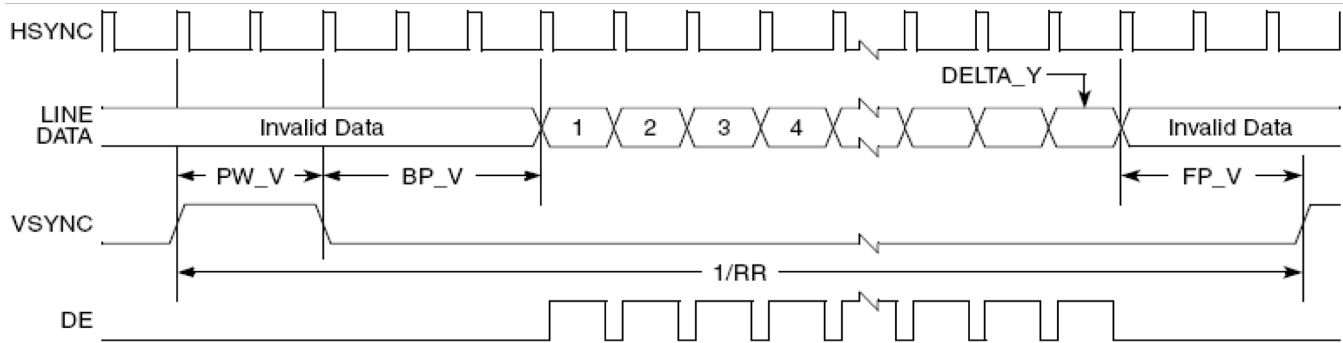


Figure 12-50. Vertical Sync Signals

By default, the DIU outputs (data and sync signals) switch at same time with the pixel clock rising edge. To provide flexibility in meeting the timing requirements of different LCD display drivers, the user can perform minor tuning of the timing of the pixel clock found on the DIU_CLK pin relative to all the other DIU signals (DIU_LD[23:0], DIU_VSYNC, DIU_HSYNC, DIU_DE). This phase tuning is done using programmable parameters in the DCCR register, specifically DCCR[CLK_INV] and DCCR[DLY_NUM] (refer to the system clock chapter). Phase tuning using CLK_INV and DLY_NUM does not change the pixel clock frequency or duty cycle.

12.4.13.1 Refresh Rate

The refresh rate (or frame rate) is the number of times that the display is updated in a second. It can be calculated from the timing parameters using this formula:

$$rr = \frac{\text{pix_clk}}{(\text{delta_x} + \text{fp_h} + \text{pw_h} + \text{bp_h}) \times (\text{delta_y} + \text{fp_v} + \text{pw_v} + \text{bp_v})}$$

Because the user probably have a set target refresh rate (RR), the PIX_CLK value has been set already and the DELTA_X and DELTA_Y values are determined exactly by the panel used. The rest of the parameters in this equation must be chosen to approach the desired refresh rate while complying with the requirements established in the panel's data sheet for the front and back porches.

12.5 Initialization/Application Information

12.5.1 DIU Initialization

The procedure to bring up the DIU out of hardware reset state, start executing data processing, and display functions is as follows:

1. Hardware reset.
2. Configure I/O function multiplexing and drive strength for DIU related pins.
3. Program the display timing signal generation related registers. Failing to set appropriate values for the display timing parameters may result in damage to the display.

4. Program DIU pixel clock divide ratio and turn on the clock in the system clock module. Enable pixel clock inversion or the programmable pixel clock delay if necessary so that the interface AC timing requirement can be met.
5. Program the DIU PLUT register and the DRAM controller priority manager as appropriate so that the DIU priority can be escalated dynamically, to make sure no buffer underrun would be hit.
6. Prepare the palette, gamma tables, cursor bitmap in memory (DDR DRAM) and set the pointers to them. This step is not strictly required because the user might be using it in an application without palette, gamma, or cursor. However, it's recommended to load zero contents in this case so that the internal SRAM can be initialized. The gamma table is always in use except in mode 3.
7. Prepare the area descriptors in memory (DDR DRAM) and set the pointers to them. This step is also not strictly required because the default background colors can be displayed or a mode that does not require area descriptors (mode 4) may be selected.
8. Program the INT_MASK register and enable the interrupts needed for the application (by default, all interrupts are disabled after hardware reset).
9. Configure the WB_MEM_ADDR register if the operating mode is mode 2 or 3.
10. Set the operating mode (by configuring the DIU_MODE register) to turn on the DIU.

12.5.2 Controlling DIU Planes after the DIU is Enabled

The DIU is initialized by correctly configuring its registers before enabling the DIU by setting the DIU_MODE to a legal, non-zero value. The DIU supports up to 3 planes which are:

- activated by setting the corresponding DESC_n register to a non-zero value
- deactivated by setting the corresponding DESC_n register to 0x0000_0000

12.5.3 Synchronize With the Host

Three interrupt status bits are defined in the DIU for synchronization purpose. They are VSYNC, LS_BF_VS, and VSYNC_WB.

If enabled the VSYNC status bit is always asserted first cycle while the VSYNC pulse is active. With this interrupt the host can always observe the beginning of a new frame. Beside this, another interrupt, LS_BF_VS (lines before vsync) can be used to set a deadline for the host to program the DIU for the next frame. This interrupt is asserted user specified lines (set by the LS_BF_VS threshold, [Figure 12-17](#)) before the vertical front porch (FP_V).

[Figure 12-51](#) shows a timing diagram on how these two interrupts can be used to synchronize the host and the DIU. At the end of each frame the DIU starts processing the next frame by first loading the necessary AD, palette, cursor, and gamma information from external memory pointed to by its internal register values. All of this takes place in the time window marked by the two dotted blue lines in [Figure 12-51](#). The host needs to make sure the proper data is in external memory and proper address values are programmed into the DIU's registers before this window. Reprogramming the palette, gamma, cursor, or AD pointers in this time window is blocked. For this reason the host should use the LS_BF_VS interrupt to start setting up the DIU for the next frame, while the LS_BF_VS threshold should be set to trigger the interrupt before

FP_V pulse of the current frame (set it to greater than 0) or the host does not have enough time to properly reprogram the DIU. Use the VSYNC_WB interrupt for mode 3.

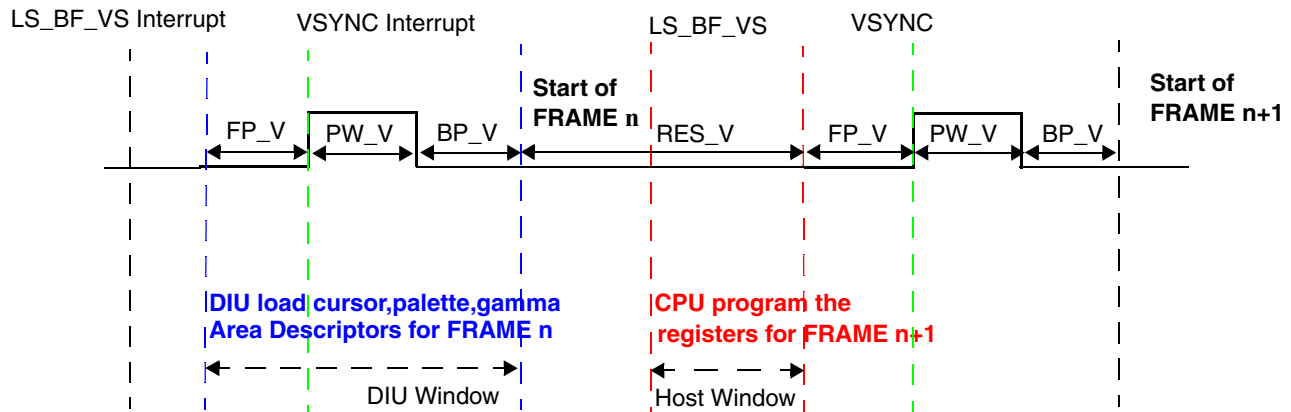


Figure 12-51. Synchronize the Host and the DIU

For operating mode with write back operation, a VSYNC_WB interrupt can be used to find the end of a write back frame. To enable the DIU to work on the next write back frame, the user need to reprogram the DESC_1/DESC_2/DESC_3 pointers for related planes, and the WB_MEM_ADDR register after the VSYNC_WB interrupt is detected.

12.5.4 Recover From Parameter Error

A parameter error exception occurs under the conditions detailed in [Table 12-38](#). If enabled, a PARERR interrupt is issued to the host. On reception of a PARERR interrupt, the user should turn off the DIU, correct the wrong parameters, and then turn it on again (by programming the DIU_MODE register). The DIU is initialized internally.

12.5.5 Recover From Underrun Error

In a heavily loaded system, the DIU may hit buffer underrun error because of long DRAM access latency. To eliminate buffer underrun issue the user should program the MPC5121E DRAM controller priority manager so that the DIU has higher priority compare to the other masters, or program the priority manager and the DIU PLUT register to enable dynamic DIU priority escalation so that the DIU priority can be high enough while it's buffer filling is low.

If a single buffer underrun occurs and it's short, the DIU may repeat the pixel before the underrun and then recover automatically when the underrun is gone to minimize impact to the display. Slight underrun error(s) do not propergate between frames, but if underrun error takes place frequently the user should turn off the DIU, escalate the DIU priority if necessary, and then turn it on again so that it can display normally.

Chapter 13

DRAM Controller

13.1 Introduction

The DDR DRAM controller is a multi-port DRAM controller (5 ports). It supports Mobile-DDR¹, DDR-1, and DDR-2 memories.

A block diagram of the multi-port DRAM controller is given in [Figure 13-1](#).

13.1.1 Overview

The DRAM controller is a multi-port controller that listens to incoming requests on the five incoming busses and decides on each rising clock edge what command needs to be sent to the DRAM.

Each incoming bus is a 64-bit bus. The five incoming busses are:

- Bus 0: the DIU , VIU
- Bus 1: the Power Architecture e300 core, PCI controller
- Bus 2: AXE audio processor
- Bus 3: MBX graphics accelerator²
- Bus 4: DMA, USB, FEC, SATA

The block supports connection of 1 DRAM rank (1 chip select) and supports the three mayor classes of DRAM:

- Mobile-DDR (LPDDR)
- DDR1
- DDR2

It supports these memories in 16-bit or 32-bit wide configurations.

The DRAM controller listens to the incoming requests to the five busses in parallel and then sends commands to the DRAM from the highest priority bus at the current time the, while the DRAM is ready to receive the command from this particular bus. If the DRAM is blocked because it needs to meet a timing requirement, the controller sends a command from a bus where there is no blockage.

For example, suppose bus one has an incoming request on priority four, and it hits in bank 1 and the page is not open (the bank needs a precharge+activate command before the request can be serviced). Bus two has an incoming request on priority five, it hits in bank two and the correct page is already open. In this case, the DRAM controller accepts the bus two request first. While its reading from the appropriate bank,

1. JEDEC standard calls these LPDDR. Most DRAM vendors call them Mobile-DDR.

2. MBX only available on MPC5121e, not on MPC5123

it issues the active+precharge command for the bus one request. Because the DRAM controller sees it cannot issue the read for the bus one request (the bank needs precharge + activate), it takes the bus two request first. Because it can issue the read, the correct page is open. During this, it issues the precharge + activate for the bus 1 request in the background. This request does not suffer from the bus two request being serviced first.

The embedded priority manager determines the relative priority of each bus, and this is used by the DRAM controller to determine which requests are more urgent.

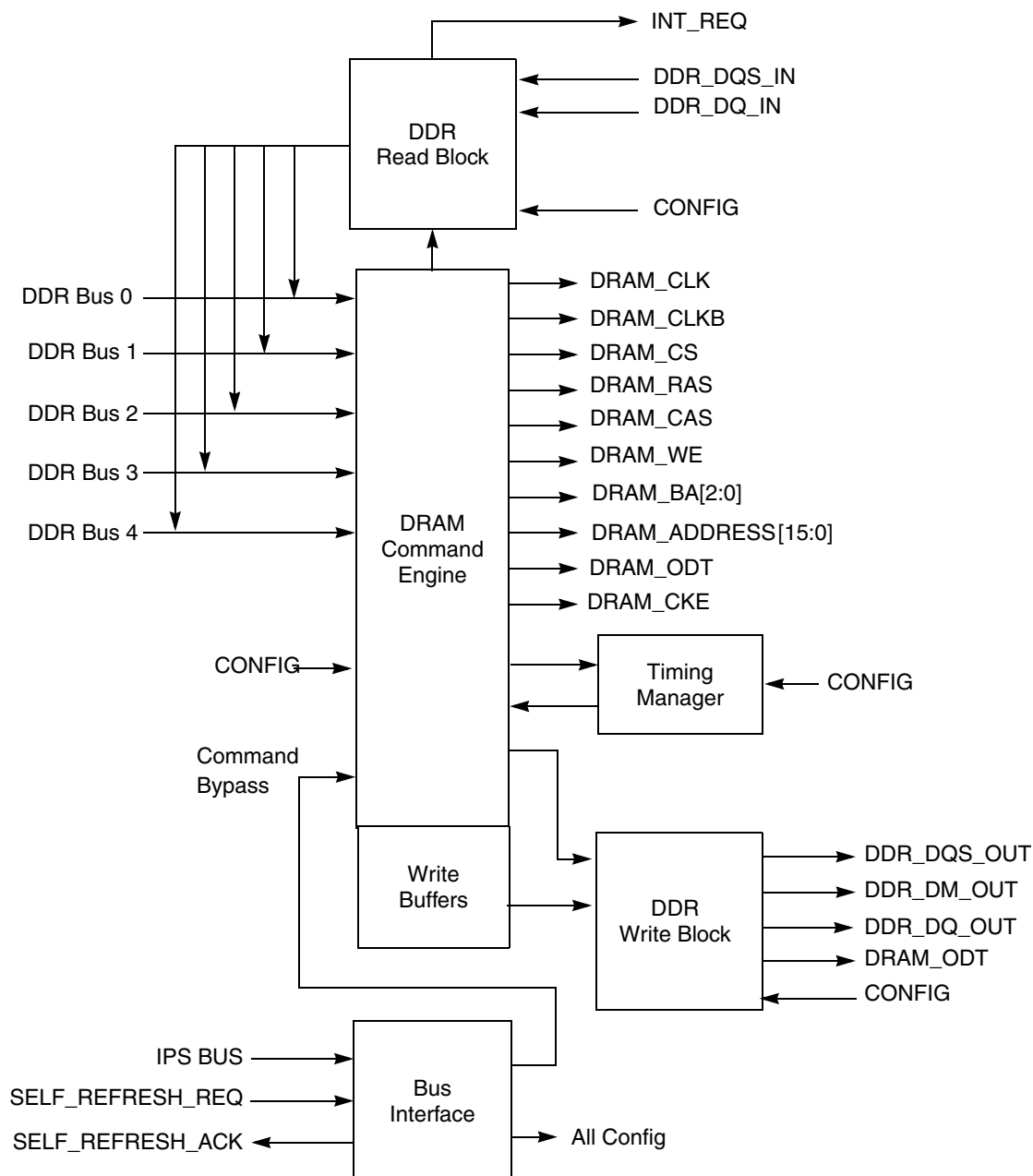


Figure 13-1. Block Diagram of the Multi-port DRAM Controller

13.2 Features

- Supports CAS latency of 2, 3, and 4 clock cycles.
- Master busses
 - Five incoming master busses
 - Supports 16-byte and 32-byte bursts
 - Supports byte enables
 - Supports 4-bit priority signal for each bus.
- Arbitration protocol
 - Inside the arbiter block, there are a total of six different arbiters that each take out the highest priority request in a certain class. All the arbiters are DRAM state aware, meaning they disregard requests that cannot be sent to the DRAM because of DRAM timing limitations.
 - Arbiter - 1: Looks for highest priority read command
 - Arbiter - 2: Looks for highest priority write command
 - Arbiter - 3: Looks for highest priority activate-for-read command
 - Arbiter - 4: Looks for highest priority activate-for-write command
 - Arbiter - 5: Looks for highest priority precharge-for-read command
 - Arbiter - 6: Looks for highest priority precharge-for-write command
 - After the first prioritization, the next round of arbitrating between the different arbiters is done. A fixed-priority schema is followed:
 - Read and write commands have highest priority
 - Activate has next-highest priority
 - Precharge has lowest priority
 - The DRAM is in read or write mode. In read mode, reads have priority over writes. In write mode, writes have priority over read.
 - DRAM only switches from read to write mode or vice-versa if:
 - A high-priority write is found, and the write buffer is full.
 - A high-priority read is found.
 - The device is in read mode, but no more reads pending
 - The device is in write mode, but no more writes pending.
- Write buffer contains five 32-byte entries.
- Supports 16-wide and 32-wide DDR1/DDR2 and Mobile-DDR DRAM devices
- Controller supports one chip select, 8-bank DRAM system
- Supports dynamic on-die termination in the host device and in the DRAM.

13.3 Memory Map and Register Definition

13.3.1 Memory Map

Table 13-1. DRAM Controller Memory Map

Offset or Address	Register	Access
(base) + 0	DDR_SYS_CONFIG	R/W
(base) + 4	DDR_TIME_CONFIG0	R/W
(base) + 8	DDR_TIME_CONFIG1	R/W
(base) + 0xC	DDR_TIME_CONFIG2	R/W
(base) + 0x10	DDR_COMMAND	R/W
(base) + 0x14	DDR_COMPACT_COMMAND	R/W
(base) + 0x18	SELF_REFRESH_CMD_0	R/W
(base) + 0x1C	SELF_REFRESH_CMD_1	R/W
(base) + 0x20	SELF_REFRESH_CMD_2	R/W
(base) + 0x24	SELF_REFRESH_CMD_3	R/W
(base) + 0x28	SELF_REFRESH_CMD_4	R/W
(base) + 0x2C	SELF_REFRESH_CMD_5	R/W
(base) + 0x30	SELF_REFRESH_CMD_6	R/W
(base) + 0x34	SELF_REFRESH_CMD_7	R/W
(base) + 0x38	DQS config offset count	R/W
(base) + 0x3C	DQS config offset time	R/W
(base) + 0x40	DQS delay status	R

13.3.2 Register Descriptions

13.3.2.1 DDR System Configuration Register

Offset: 0x0000 Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RST_B	CKE	CLK ON	CMD MODE	DRAM_ROW_SELECT			DRAM_BANK_SELECT				READ_TEST		SELF REF EN	16BIT MODE	RDLY [3]
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDLY[2:0]			HALF DQS DLY	QUART DQS DLY	WDLY[2:0]			EARLY ODT	ON DIE TERMINATE			FIFO OVERFLOW	FIFO UNDERFLOW	FIFO OVERFLOW	FIFO UNDERFLOW
W											FIFO OVERFLOW CLEAR	FIFO UNDERFLOW CLEAR				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<div></div> = Unimplemented or Reserved																

□ = Unimplemented or Reserved

Figure 13-2. DDR System Configuration Register (Sheet 1 of 3)
(Register is repeated for reference.)

Table 13-2. DDR System Configuration Register Field Descriptions

Field	Description
RST_B	DRAM controller soft reset. When this bit is 0, the DRAM controller is in the reset state. When this bit is 1, the DRAM controller is out of reset. The bit controls the reset to the internal state machines. The configuration registers are reset by the resets from the hardware reset block, not by this bit.
CKE	Value on the DRAM CKE pin. For functional operation, this needs to be high. During power-down, value can be low.
CLK ON	When this bit is 1, the DRAM clock is running. When this bit is 0, the DRAM clock is stopped
CMD MODE	When this bit is 0, the DRAM controller is in normal operation. When this bit is 1, the DRAM controller is in command mode and does not respond to requests on the incoming busses. Command mode is used for DRAM initialization and to switch the DRAM into and out of the different power-down and self-refresh modes.
DRAM_ROW_SELECT and DRAM_BANK_SELECT	These fields control the multiplexing of the bus address to the DRAM bank and row address. Table 13-6 and Table 13-7 give the details. DRAM column address depends on 16-bit mode bit, and relationship is given in Table 13-6 .
READ_TEST	These fields are for production test. Don't use.
SELFREFEN	Self-refresh enable. When this bit is 1, the DRAM controller autonomously enters and exits the self-refresh using the self-refresh command registers when requested by the PMC. When the bit is 0, the transition is blocked.

DRAM Controller

Offset: 0x0000 Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RST_B	CKE	CLK_ON	CMD_MODE	DRAM_ROW_SELECT	DRAM_BANK_SELECT						READ_TEST	SELF_REF_EN	16BIT_MODE	RDLY[3]	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDLY[2:0]			HALF_DQS_DLY	QUART_DQS_DLY	WDLY[2:0]			EARLY_ODT	ON_DIE_TERMINATE			FIFO_OVERFLOW	FIFO_UNDERFLOW	FIFO_OVERFLOW	FIFO_UNDERFLOW
W											FIFO_OVERFLOW_CLEAR	FIFO_UNDERFLOW_CLEAR				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<div></div> = Unimplemented or Reserved																

Figure 13-2. DDR System Configuration Register (Sheet 2 of 3)
(Register is repeated for reference.)

Table 13-2. DDR System Configuration Register Field Descriptions (continued)

Field	Description
16-BIT MODE	When this bit is set, the DRAM controller assumes a 16-bit wide memory is used. When this bit is cleared, a 32-bit wide memory is assumed. Note: This does not configure the pins for 16-bit mode. That must be done in the pin configuration.
RDLY[3:0]	This field controls the expected delay between sending a read command to the DRAM and receiving the read data from the DRAM. RDLY, HALF DQS DLY, and QUART DQS delay together to code for t_{DQSEN} . The t_{DQSEN} is the delay between the read command and when the internal DQS enable goes high. See Figure 13-3 . Timing is internally compensated, and is referred to timing at the device pins. t_{DQSEN} should be selected so the L-H transition of DQS enable is always in the preamble of the DQS input of the READ command. Required t_{DQSEN} value depends on the CAS latency (CL), the distance between the DRAM and the device, and the type of DRAM used. Table 13-3 gives the detail on programming t_{DQSEN} .
HALF DQS DLY	This field is an extra field to control the expected read delay between issuing the read command and getting read data from the DRAM. This field offers 1/2 CSB clock granularity when programming the delay. See description of field RDLY for details
QUART DQS DLY	This field is an extra field to control the expected read delay between issuing the read command, and getting read data from the DRAM. This field offers 1/4 csb clock granularity when programming the delay. - see description of field RDLY for details rdly, hald
WDLY[2:0]	This field controls the write latency (WL) for write commands. See Table 13-4 .
EARLY ODT	This bit needs to be set if write latency is 1 (wdly[2:0] = 001) and on die termination is used with DDR2 DRAM. It makes sure the DRAM controller asserts the ODT signal going to the DRAM one clock ahead of issuing the write command.

Offset: 0x0000Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RST_B	CKE	CLK ON	CMD MOD E	DRAM_ROW_SELECT	DRAM_BANK_SELECT						READ_TEST		SELF REF EN	16BIT MOD E	RDLY [3]
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDLY[2:0]			HALF DQS DLY	QUART DQS DLY	WDLY[2:0]			EARLY ODT	ON DIE TERMINATE			FIFO OVPEN D	FIFO UV PEN D	FIFO OVEN	FIFO UV EN
W											FIFO OVCLEAR	FIFO UV CLEAR				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<div></div> = Unimplemented or Reserved																


 = Unimplemented or Reserved

Figure 13-2. DDR System Configuration Register (Sheet 3 of 3)
(Register is repeated for reference.)

Table 13-2. DDR System Configuration Register Field Descriptions (continued)

Field	Description
ON DIE TERMINATE	If this bit is 1, the internal pads generates on-die termination during read. If the bit is clear, there is no on die termination. This bit controls ODT in the controller. The ODT in the DRAM is controlled via the DRAM internal configuration registers registers. Please consult DRAM data sheet for it.
FIFO OV CLEAR FIFO UV CLEAR FIFO OV PENDING FIFO UV PENDING FIFO OV EN FIFO UV EN	<p>These bits control the interrupt generation by the DRAM controller. The DRAM controller has two interrupts: FIFO OV pending and FIFO UV pending. These interrupts are set on overflow or underflow of the FIFO in the read block. When a read command is sent to the DRAM, it is entered into a FIFO. The DRAM is expected to answer by sending back the read data with some up and down edges on the DQS lines (the DQS strobes) used to clock the data. The DRAM controller clocks the read data with the DQS strobes supplied by the DRAM and retrieves the read command from the FIFO after receiving the correct number of read strobes. When the read data strobes returned by the DRAM do not match the expectations of the controller, the FIFO may underflow (if too many clocks are coming back from the DRAM) or overflow (if not enough clocks are coming back). These underflows and overflows are basically the result of problems with the DRAM interface or incorrect parameter settings in the controller or the DRAM. Care has been taken during the design of the DRAM controller not to enter a hang-up state when this occurs. However, read data is corrupt and CPU is informed via the FIFO overflow and FIFO underflow interrupts. The issue is also discussed in Section Bus Interface on page 13-28.</p> <ul style="list-style-type: none"> FIFO_OV_PENDING and FIFO_UV_PENDING signal to the CPU if an overflow or underflow interrupt is pending FIFO_OV_EN and FIFO_UV_EN bits are interrupt enable bits. If the pending + enable bit is set at the same time, the interrupt is sent to the e300 CPU. FIFO_OV_CLEAR and FIFO_UV_CLEAR are clear bits. Writing a 1 to either of these clears the pending interrupt.

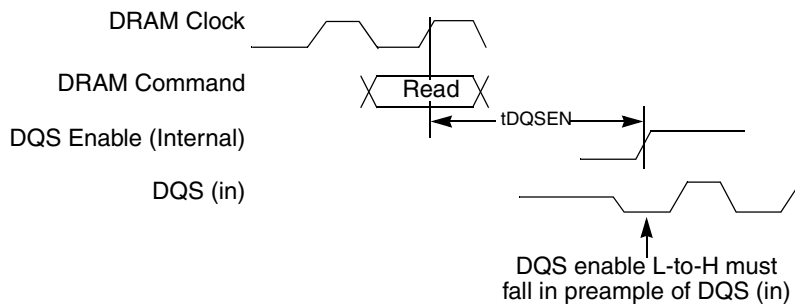


Figure 13-3. t_{DQSEN}

Table 13-3. Programming t_{DQSEN}

{rdly[3:0],half_dqs_dly,quart_dqs_dly}	t_{DQSEN} (CSB Clock Periods)
1000 0 0	0.5
1000 0 1	0.75
1000 1 0	1.0
1000 1 1	1.25
0100 0 0	1.5
0100 0 1	1.75
0100 1 0	2.0
0100 1 1	2.25
0010 0 0	2.5
0010 0 1	2.75
0010 1 0	3.0
0010 1 1	3.25
0001 0 0	3.5
0001 0 1	3.75
0001 1 0	4.0
0001 1 1	4.25
0000 0 0	4.5
0000 0 1	4.75
0000 1 0	5.0
0000 1 1	5.25

Table 13-4. Write Latency

WDLY[2:0]	Write Latency (CSB Clocks)
100	4
011	3

Table 13-4. Write Latency (continued)

WDLY[2:0]	Write Latency (CSB Clocks)
010	2
001	1

Table 13-5. Number of DRAM Banks Addressed and Mapping of Address to DRAM Bank Address

DRAM_BANK_SELECT	Number of Banks	DRAM_BANK
0	4	DRAM_BANK[1:0] = address[11:10]
1	4	DRAM_BANK[1:0] = address[12:11]
2	8	DRAM_BANK[2:0] = address[13:11]
3	4	DRAM_BANK[1:0] = address[13:12]
4	8	DRAM_BANK[2:0] = address[14:12]
5	4	DRAM_BANK[1:0] = address[14:13]
6	8	DRAM_BANK[2:0] = address[15:13]
7	4	DRAM_BANK[1:0] = address[15:14]
8	8	DRAM_BANK[2:0] = address[16:14]
9	4	DRAM_BANK[1:0] = address[25:24]
10	8	DRAM_BANK[2:0] = address[26:24]
11	4	DRAM_BANK[1:0] = address[26:25]
12	8	DRAM_BANK[2:0] = address[27:25]
13	8	DRAM_BANK[2:0] = address[28:26]
14	8	DRAM_BANK[2:0] = address[29:27]
15	8	DRAM_BANK[2:0] = address[30:28]

Table 13-6. Mapping of Address to DRAM Column Address

16-Bit Mode	DRAM_COLUMN[12:0]
0	DRAM_COLUMN[12:0] = address[14:2]
1	DRAM_COLUMN[12:0] = address[13:1]

NOTE

In 16 bit mode (16BITMODE = 1), DDR memories with column address line 0 to 7 are not supported.

Table 13-7. Mapping of Address to DRAM Row Address

DRAM_ROW_SELECT [2:0]	DRAM_ROW[15:0]
0	DRAM_ROW[15:0] = address[25:10]
1	DRAM_ROW[15:0] = address[26:11]
2	DRAM_ROW[15:0] = address[27:12]
3	DRAM_ROW[15:0] = address[28:13]
4	DRAM_ROW[15:0] = address[29:14]
5	DRAM_ROW[15:0] = address[30:15]
6	DRAM_ROW[14:0] = address[30:16]
7	DRAM_ROW[13:0] = address[30:17]

13.3.2.2 Timing Configuration

Offset: 0x0004

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DRAM_REFRESH_TIME[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DRAM_COMMAND_TIME[7:0]								DRAM_BANK_PRE_TIME[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-4. DDR Time Config0 Register

Table 13-8. DDR Time Config0 Register Field Descriptions

Field	Description
DRAM_REFRESH_TIME[15:0]	Refresh interval of the DRAM. Program in this register the number of CSB clocks between any two refresh requests. This register should contain the maximum number of CSB clocks between two refresh requests. The average time in CSB clock periods between two refreshes to the DRAM is this number.
DRAM_COMMAND_TIME[7:0]	Time-out after sending a command to the DRAM in bypass mode. For command sent to the DRAM using the DDR_COMMAND and DDR_COMPACT_COMMAND register, the normal checking of the timing parameters is not done. Instead, any new command to the DRAM is disabled for DRAM_COMMAND_TIME[7:0] dram clock periods. This parameter needs to be programmed for the worst-case time-out.

Offset: 0x0004

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DRAM_REFRESH_TIME[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DRAM_COMMAND_TIME[7:0]								DRAM_BANK_PRE_TIME[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-4. DDR Time Config0 Register

Table 13-8. DDR Time Config0 Register Field Descriptions (continued)

Field	Description
DRAM_BANK_PRE_TIME[7:0]	Time-out. Any active bank, that has no outstanding requests, is automatically precharged by the DRAM controller after this time-out has elapsed since the last access to the bank. This time can be set short, which results in open banks being precharged quite fast to long, which results in open banks left open for long time. The value is a time count in dram clock periods.

Offset: 0x0008Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DRAM_TIME_RFC[5:0]					DRAM_TIME_WR1[4:0]					DRAM_TIME_WTR1[3:0]			DRAM_TIME_RRD[5]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DRAM_TIME_RRD[4:0]					DRAM_TIME_RC[5:0]					DRAM_TIME_RAS[4:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-5. DDR Time Config1 Register

DRAM Controller

Offset: 0x000C

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DRAM_TIME_RCD[3:0]				DRAM_TIME_FAW[4:0]				DRAM_TIME_RTW1[3:0]				DRAM_TIME_CCD[3:1]			
W																
Reset	0	0	0	0	0	0	0	0	0	0						
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DRAM_TIME_CCD[0]	DRAM_TIME_RTP[4:0]				DRAM_TIME_RP[4:0]				DRAM_TIME_RPA[4:0]						
W																
Reset	0	0	0	0	0	0	0	0					0		0	

Figure 13-6. DDR Time Config2 Register

DDR_TIME_CONFIG1 and DDR_TIME_CONFIG2 registers need to be programmed with the DDR1/DDR2 timing parameters. All times are given in clock cycles.

The timing parameters are conceived so the controller CSB clock cycles match with the Jedec DDR2 specification. To interface with DDR1 or Mobile-DDR(LPDDR), some timing parameters need not be enforced, or are calculated differently. Refer to DRAM datasheet to determine their value. The timing parameters need to be programmed in function of this DRAM requirement. [Table 13-9](#) gives the details.

Table 13-9. Timing Parameters

Timing Parameter	Controls Jedec Parameter (Jedec spec)	Formulae (All times in CSB clock periods)	Description
DRAM_TIME_RFC	t_{RFC}	$DRAM_TIME_RFC = t_{RFC}$	REFRESH to ACTIVE or REFRESH to REFRESH command interval
DRAM_TIME_RRD	t_{RRD}	$DRAM_TIME_RRD = t_{RRD}$	ACTIVE bank A to ACTIVE bank B command
DRAM_TIME_RC	t_{RC}	$DRAM_TIME_RC = t_{RC}$	ACTIVE to ACTIVE (same bank) command
DRAM_TIME_RAS	t_{RAS}	$DRAM_TIME_RAS = t_{RAS}$	ACTIVE to PRECHARGE command
DRAM_TIME_RCD	t_{RCD}	$DRAM_TIME_RCD = t_{RCD}$	ACTIVE to READ or WRITE delay
DRAM_TIME_FAW	t_{FAW}	$DRAM_TIME_FAW^1 = t_{FAW}$	4-bank activate period
DRAM_TIME_CCD	t_{CCD}	$DRAM_TIME_CCD^2 = \max(t_{CCD,2})$ (32-bit mode) $\max(t_{CCD,4})$ (16-bit mode)	CAS to CAS delay Because time is needed for data to be sent over, this time is minimum two clocks in 32-bit mode and four clocks in 16-bit mode

Table 13-9. Timing Parameters (continued)

Timing Parameter	Controls Jedec Parameter (Jedec spec)	Formulae (All times in CSB clock periods)	Description
DRAM_TIME_RTP	t_{RTP}	$DRAM_TIME_RTP^3 = t_{RTP}$ (32-bit mode, DDR2) $t_{RTP}+2$ (16-bit mode, DDR2)	Read to precharge delay. DRAM_TIME_RTP is the read-to-precharge delay and t_{RTP} is the <i>internal</i> read-to-precharge delay, hence, the difference for 16-bit mode. Figure 13-7 gives the details.
DRAM_TIME_RP	t_{RP}	$DRAM_TIME_RP = t_{RP}$	Precharge command period
DRAM_TIME_RPA	t_{RP}	$DRAM_TIME_RPA^4 = t_{RP} + 1$ (8 bank device) $DRAM_TIME_RPA = t_{RP}$ (4 bank device)	Precharge all command period
DRAM_TIME_WR1	t_{WR}	$DRAM_TIME_WR1 = WL + t_{WR} + 2$ (32-bit mode) $WL + t_{WR} + 4$ (16-bit mode)	DRAM_TIME_WR1 is the write recovery time, measured in clocks between write command and precharge command. For this reason, WL (the write latency) and the length of the actual write (2 or 4) need to be added to t_{WR} . Figure 13-8 gives the details.
DRAM_TIME_WTR1	t_{WTR}	$DRAM_TIME_WTR1 = WL + t_{WTR} + 2$ (32-bit mode) $WL + t_{WTR} + 4$ (16-bit mode)	DRAM_TIME_WTR1 is the write to read time, measured in clocks between write command and read command. For this reason, WL (the write latency) and the length of the actual write (2 or 4) need to be added to t_{WTR} . Figure 13-9 gives the details
DRAM_TIME_RTW1	—	$DRAM_TIME_RTW1 = CL - WL + 2 + t_{BTA}$ (32-bit) $CL - WL + 4 + t_{BTA}$ (16-bit)	DRAM_TIME_RTW1 is the read-to-write time, measured in clocks between the read and write command. There is no limitation on the DRAM on how to set this parameter. The parameter should be set such that there is no contention on the DQ data bus when switching from read to write. Equation given at left tries to come up with a formulae that defines the minimum value of DRAM_TIME_RTW1 to avoid contention. CL is the cas latency, WL is the write latency, and t_{BTA} is the bus turn-around time. t_{BTA} is the minimum dead time that needs to be put on the bus between the 5121e driving the bus and the DRAM driving the bus to take into account the transit delay on the PCB, the pad delay, the DRAM skew, and the on-chip delay.

¹ For DRAMs that do not need this check, set equal to $4 * t_{RPD}$

² For DDR1 and Mobile-DDR t_{CCD} is 2 for 32-bit operation, 4 for 16-bit operation.

³ For DDR1 and Mobile-DDR mode, t_{RTP} is not explicitly given. It is equal to 4 for 16-bit mode, equal to 2 for 32-bit mode.

⁴ This timing parameter controls precharge all command period duration. The equations shown are the Jedec definition of the t_{RPA} . Some DRAM vendors do not follow Jedec on this, and list t_{RPA} directly. In this case, set $DRAM_TIME_RPA = t_{RPA}$.

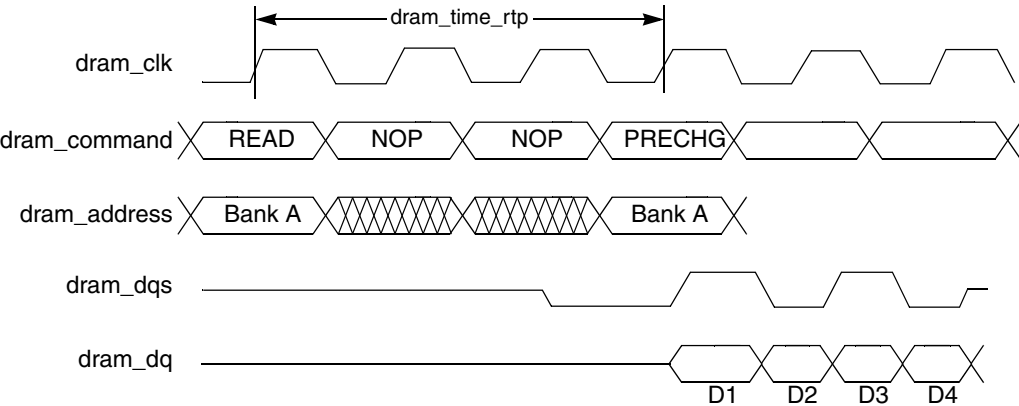


Figure 13-7. Read to Precharge Timing Diagram

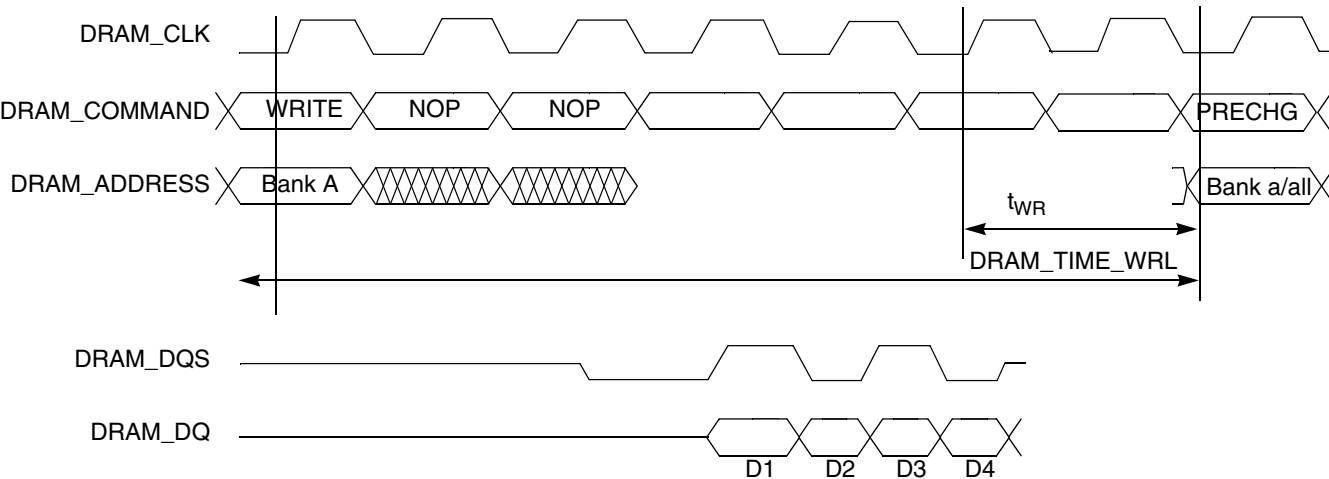


Figure 13-8. Write to Precharge Timing Diagram

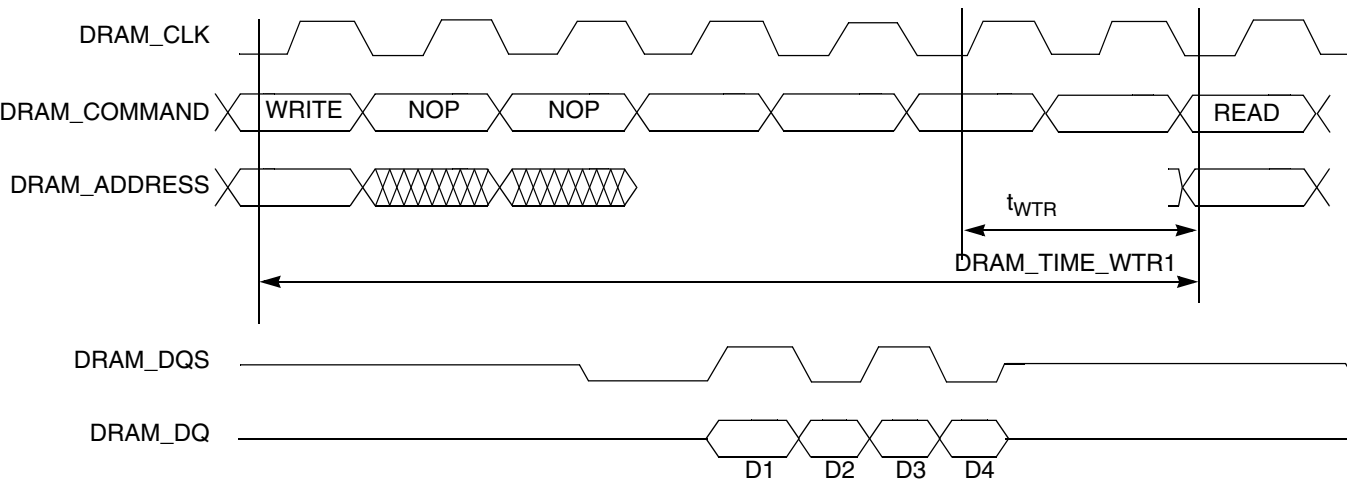


Figure 13-9. Write to Read Timing Diagram

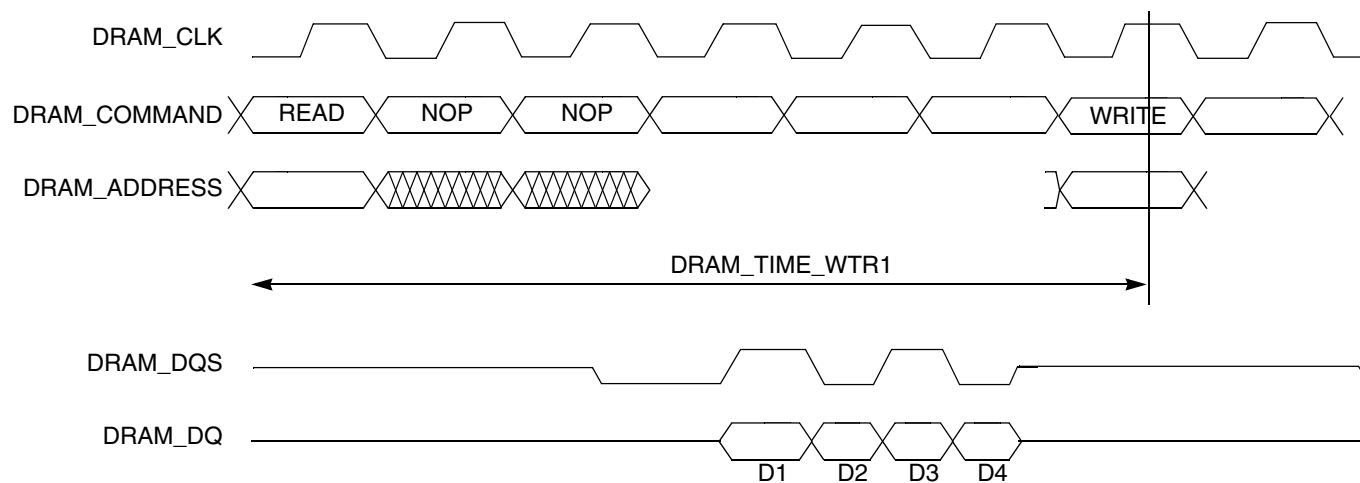


Figure 13-10. Read to Write Timing Diagram

13.3.2.3 Command Register

Offset: 0x0010

Access: Write

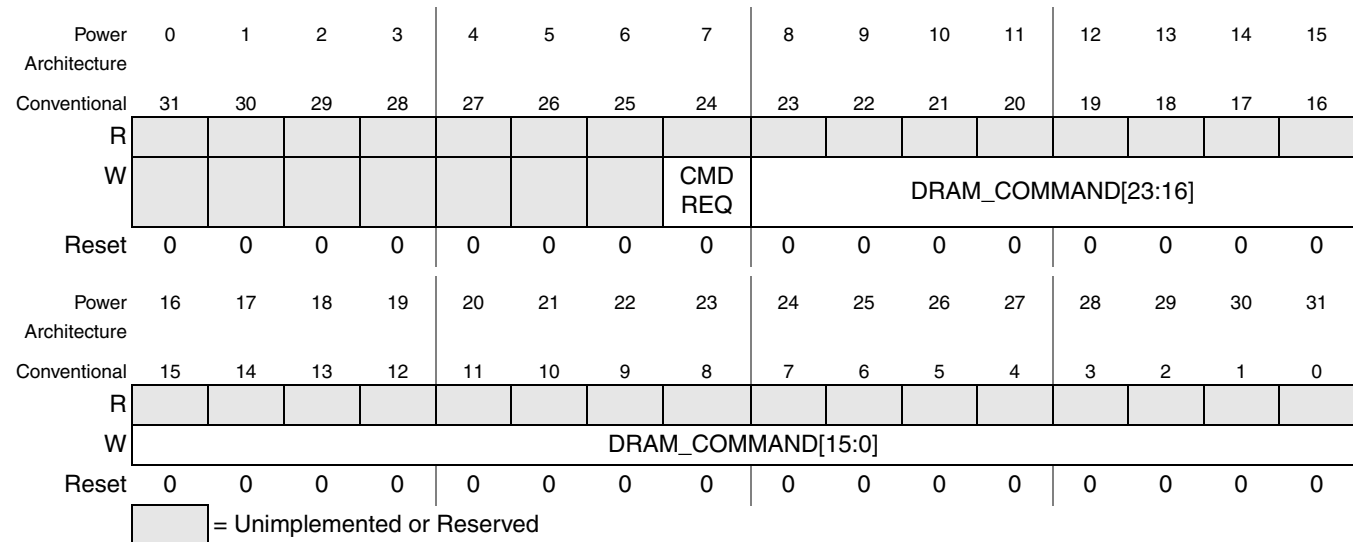


Figure 13-11. DRAM Command Register

Table 13-10. DRAM Command Register Field Descriptions

Field	Description
CMD REQ DRAM_COMMAND	<p>The DRAM command register gives the option to send commands directly to the DRAM. This register only operates when the command mode bit (bit 29) is set in the DDR_SYS_CONFIG register.</p> <p>When this bit is set and a 1 is written to bit 24 (CMD req) of this register, the value written to bits [23:0] of this register is output on the DRAM address group with following mapping:</p> <ul style="list-style-type: none"> • DRAM_ADDRESS[14:0] = DRAM_COMMAND[15:0] • DRAM_ADDRESS[15] = 0 • if(DRAM_COMMAND[15] == 1) turn off CKE DRAM attribute bit¹ • DRAM_BA[2:0] = DRAM_COMMAND[18:16] • DRAM_WEB = DRAM_COMMAND[19] • DRAM_CAS = DRAM_COMMAND[20] • DRAM_RAS = DRAM_COMMAND[21] • DRAM_CS = DRAM_COMMAND[22] <p>Note: The intended use of the command interface is to initialize the DRAM and to put the DRAM into or out of the self-refresh and power-down modes.</p>

¹ If the CKE is turned off, it is turned off the clock cycle after the command is written to the DRAM.

13.3.2.4 Compact Command Register

Offset: 0x0014

Access: Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	DRAM_COMPACT_COMMAND[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-12. Compact Command Register

Table 13-11. Compact Command Register Field Descriptions

Field	Description
DRAM_COMPACT_COMMAND[15:0]	The compact command register gives the option to sent commands to the DRAM using 16-bit writes. See Table 13-12

Table 13-12. Compact Command Register Options

compact_command[15:14]	
00	Write DRAM attributes and wait (wait time = wait time till next command) Wait is executed after writing attributes. <ul style="list-style-type: none"> • CKE = COMPACT_COMMAND[13] • Self Ref En = COMPACT_COMMAND[12] • CLK ON = COMPACT_COMMAND[11] • CMD MODE = COMPACT_COMMAND[10] • If(compact_command[7] == 1'b1) • Wait time = (compact_command[6:0] * 512) dram clock periods Or <ul style="list-style-type: none"> • Wait time = (compact_command[6:0] * 32) dram clock periods

Table 13-12. Compact Command Register Options

compact_command[15:14]	
01	DRAM command <ul style="list-style-type: none"> • DRAM_CS = COMPACT_COMMAND[12] • DRAM_RAS = COMPACT_COMMAND[11] • DRAM_CAS = COMPACT_COMMAND[10] • DRAM_WEB = COMPACT_COMMAND[9] • DRAM_BA[2:0] = COMPACT_COMMAND[8:6] • DRAM_ADDRESS[10] = COMPACT_COMMAND[5] • if(COMPACT_COMMAND[4] == 1'b1) turn off CKE DRAM attribute bit¹
1x	DRAM set mode registers <ul style="list-style-type: none"> • DRAM_CS = 0 • DRAM_RAS = 0 • DRAM_CAS = 0 • DRAM_WEB = 0 • DRAM_ADDRESS[13] = 0 • DRAM_BA[2] = 0 • DRAM_ADDRESS[12:0] = COMPACT_COMMAND[12:0] • DRAM_ADDRESS[14:13] = COMPACT_COMMAND[14:13]

¹ CKE is turned off the clock cycle after sending the requested command to the DRAM.

The COMPACT_COMMAND register's main purpose is to be written during enter/exit of self-refresh (the auto-sequencer). This is described in the following section.

The compact command register allows three types of actions to be executed:

- Write DRAM attributes and wait. Wait is executed after updating the DRAM attributes. It is possible to update the CKE bit, the self-refresh enable, the CLK configuration (on/off), and the CMD mode setting.
 - If, during the time the wait is executed, another command is written to the CompactCommand register, this write is delayed until the wait is over.

During this time, the peripheral bus and all busses connected to it block and are not able to process any other read or write.
- Write a command to DRAM without controlling the address. In this mode, it is possible to send refresh, activate, and precharge commands to the DRAM
- Write a DRAM mode register.

13.3.2.5 Enter/Exit Self-Refresh Registers

Offset: 0x0018

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD0[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 13-13. Self-Refresh Command 0 Register

Offset: 0x001C

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD1[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 13-14. Self-Refresh Command 1 Register

DRAM Controller

Offset: 0x0020

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-15. Self-Refresh Command 2 Register

Offset: 0x0024

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD3[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-16. Self-Refresh Command 3 Register

Offset: 0x0028

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD4[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-17. Self-Refresh Command 4 Register

Offset: 0x002C

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD5[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-18. Self-Refresh Command 5 Register

Offset: 0x0030

Access: Read/Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD6[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-19. Self-Refresh Command 6 Register

Offset: 0x0034

Access: Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SELF_REFRESH_CMD7[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-20. Self-Refresh Command 7 Register

The self-refresh command registers contain the commands sent to the DRAM when a self-refresh request is given. [Figure 13-21](#) gives the details.



Figure 13-21. Enter/exit Self-Refresh Command Protocol

When the DRAM controller sees a low-to-high transition on the incoming self-refresh REQ signal coming from the PMC controller, its reaction depends on the state of the internal self-refresh EN command bit.

- If the self-refresh EN bit is set, the DRAM controller writes self-refresh CMD[0:3] registers, starting with reg 0 and ending with reg 3, to the compact command register. After the last register has been written and its wait time has expired (if any), it pulls high the self-refresh ACK signal to the PMC to acknowledge the enter of self-refresh mode.
- If the self-refresh EN bit is clear, the DRAM controller does not react to the request and keeps self-refresh ACK signal low.

When the DRAM controller sees a high-to-low transition on the incoming self-refresh REQ signal coming from the PMC controller, its reaction is similar and depends on the state of the internal self-refresh EN command bit again.

- If the self-refresh EN bit is set, the DRAM controller writes self-refresh CMD[4:7] registers, starting with reg four and ending with reg seven, to the compact command register. After the last register has been written and its wait time has expired (if any), it pulls low the self-refresh ACK signal to the PMC to acknowledge the exit of the self-refresh mode.
- If the self-refresh EN bit is clear, the DRAM controller does not react to the request and keeps self-refresh ACK signal high.

13.3.2.6 DQS Config Offset Count and DQS Config Offset Time

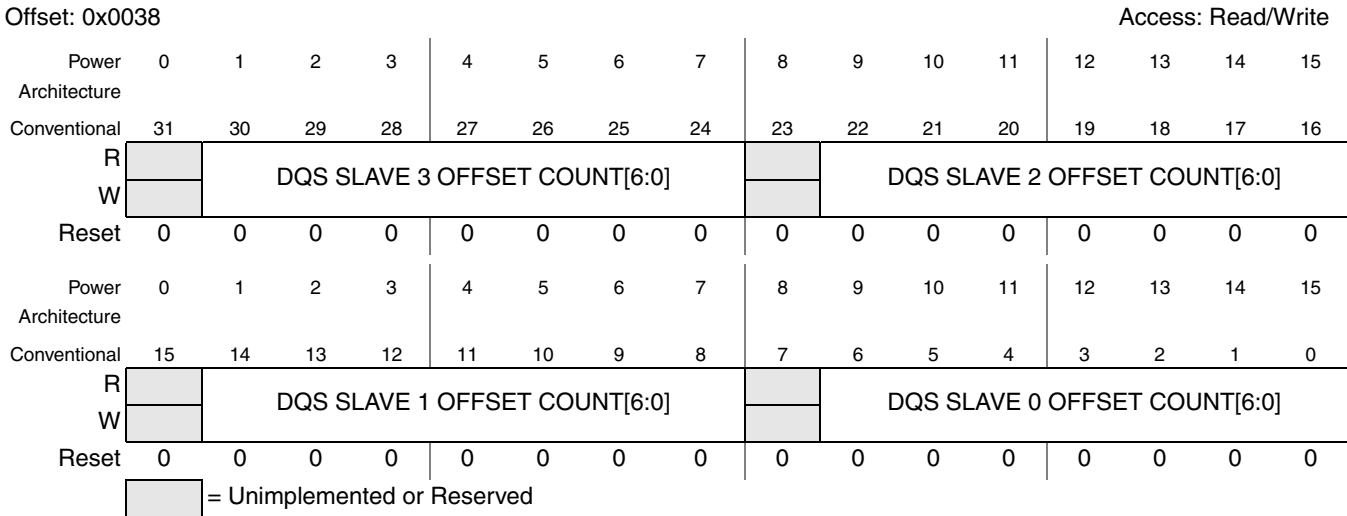


Figure 13-22. DQS Config Offset Count Register

Table 13-13. Compact Command Register Field Descriptions

Field	Description
DQS_SLAVE_[3:0]_OFFSET_COUNT	There is a separate field for each DQS input to the controller. These fields code for an offset counted in elemental gate delay increments applied to each DQS slave. The number is a two-complement number that can be positive and negative. This register can be used to compensate systematic delay shift in the DRAM controller due to processing. Leave this register all-zero, unless Freescale issues a report giving a different value.

Offset: 0x003C

Access: Read/Write

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					DQS SLAVE 3 OFFSET TIME[4:0]								DQS SLAVE 2 OFFSET TIME[4:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					DQS SLAVE 1 OFFSET TIME[4:0]								DQS SLAVE 0 OFFSET TIME[4:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 13-23. DQS Config Offset Time Register

Table 13-14. Compact Command Register Field Descriptions

Field	Description
DQS_SLAVE_[3:0]_OFFSET_TIME	<p>There is a separate field for each DQS input to the controller. These fields code for an offset counted in time units.</p> <p>This register can be used to advance or delay the read strobe. Negative values advance the read strobe, positive values retard the read strobe.</p> <p>Time delay coded = <field value (2-complement)> * Tdram-clock/256.</p> <p>The applied offset range for a 200 MHz clock is approximately ± 290 pS.</p>

13.3.2.7 DQS Delay Status

Offset: 0x0040

Access: Write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					DQS MASTER COUNT 2[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					DQS MASTER COUNT 1[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 13-24. DQS Delay Status Register

Table 13-15. DQS Delay Status Register Field Descriptions

Field	Description
DQS MASTER COUNT 2	Delay count output by the controller for the first DQS master to code for 1/4 CSB clock delay
DQS MASTER COUNT 1	Delay count output by the controller for the second DQS master to code for 1/4 CSB clock delay

13.4 Functional Description

The DRAM controller is a multi-port DRAM controller. It listens to incoming requests on multiple busses and decides on each rising clock edge what command needs to be sent to the DRAM.

A block diagram is given in [Figure 13-1](#). The major blocks of the DRAM controller are described below.

13.4.1 Interfacing with the DRAM

13.4.1.1 Connecting the DRAM

- 32-bit DRAM systems need to be connected to all DQ, DM, DQS lines.
- 16-bit DRAM systems need to be connected to the low order bits of the data bus. (DQ[15:0], DQS[1:0] and DM[1:0])
- row/column address pins need to be connected starting bit [0] and ending with the highest order DRAM bit. Leave MSB's unconnected if the DRAM has less address pins than the controller.
- DRAM bank address pins need to be connected starting bit [0] and ending with the highest order bank address bit. Leave MSB unconnected if the DRAM has less bank address pins than the controller.

13.4.2 Programming DRAM Device Internal Configuration Register

- Set burst type to sequential
- Burst length is always 16-byte. Means 4-beat bursts in a 32-bit system, 8-beat burst in a 16-bit system
- Set CAS latency to lowest value DRAM can tolerate at intended speed, and then set write latency and read latency accordingly.
- Set posted CAS additive latency to 0
- Controller never uses auto-precharge on read or write.
- Configure DQS operation for single-ended operation.
- Rtt and output drive strength configuration depends on electricals.

13.4.3 DRAM Command Engine

This block decides what command to send to the DRAM controller next. There are four different commands that can be sent to the DRAM to service incoming requests from the five incoming busses.

- Precharge
- Activate
- Read
- Write

On every rising clock edge, the DRAM command engine first determines with parallel logic what is highest priority pending precharge, activate, read and write command. Next, it decides which of these commands to send to the DRAM.

The arbiters that make the decisions about what command to send next to the DRAM are aware of the current state the DRAM is in. When arbitrating a command on the DRAM bus, the following information is processed:

- For each bank, if it is precharged or not
- For each incoming request, if it hits in an already active bank or not
- For each bank, if the DRAM currently can accept a precharge command to it
- For each bank, if the DRAM currently can accept an activate command to it
- For each bank, if the DRAM currently can accept a read command to it
- For each bank, if the DRAM currently can accept a write command to it

The logic keeping track of what is currently possible on each of the banks is not in the DRAM command engine. It is part of the timing manager, whose task is to signal to the DRAM command engine that commands are currently possible.

13.4.4 Write Buffer

All incoming writes are sent first to the write buffer, part of the command engine. Writes are sent to the DRAM in background, whenever possible. The DRAM tries to postpone the writes until there are no further outstanding read requests. However, when the write buffer is full, or when there is a new request

for an address already inside the write buffer, the DRAM controller writes the content of the write buffer to the DRAM.

13.4.5 Timing Manager

The timing manager consists of a bank of counters. These counters keep track of all DRAM timing parameters and signals to the DRAM command engine when a precharge, activate, read or write command is possible. This information is supplied to the DRAM command engine for each bank separately.

All timing parameters are programmable in software.

13.4.6 DRAM Read Block and DRAM Write Block

Sending a read or write command to the DRAM is a two-step process. First, the command is sent, which is done by the command engine. After some clock cycles, the data must follow.

Manipulating the read data is done by the read block. For every read command sent to the DRAM, the command engine informs the read block. Upon receiving the read command, the read block delays this to account for DRAM pipelining. Then, it receives the correct amount of data from the DRAM DQ inputs and forward this data to the correct bus.

Manipulating the write data is done by the write block. It works the same way as the read block. The command engine informs the write block of a pending write. Upon receiving the command, the write block delays this to account for DRAM pipelining. Then, it receives the relevant data from the write buffer and transmits this to the DRAM.

13.4.7 Bus Interface

The bus interface accepts a slave peripheral bus. The bus interface fulfills several functions:

- It contains all configuration registers
- It contains logic to send an error interrupt to the processor. The error interrupt is active when the FIFO overflow or FIFO underflow error condition and corresponding interrupt enable in register `DDR_SYS_CONFIG` is set. The register summary is given in [Figure 13-2](#).

The FIFO overflow and underflow flags are tied to a FIFO that keeps track of the number of DQS strobes the DRAM is expected to produce. If a read command is sent to the DRAM, the DRAM is expected to answer after producing the read data on its DQ outputs, with some edges on its DQS output used by the controller to clock the read data. If the DRAM controller produces the read strobes at an incorrect time, or produces not enough or too many read strobes, the DRAM controller may detect some error conditions because they result in an overflow or underflow of the FIFO that keeps track of the number of outstanding DQS pulses. These bits do not detect timing configuration errors. Underflows and overflows signaled by the read FIFO point to following possible error sources:

- Incorrect configuration of the DRAM. Burst length set incorrectly
- Incorrect configuration of the DRAM controller.
 - Incorrect RDLY

- Incorrect HALF_DQS_DLY
 - Incorrect QUART_DQS_DLY
 - Incorrect DRAM timing parameters or mis-match between various settings.
- Problems with the electrical connections between the DRAM controller and the DRAM
- It contains a bypass path to send commands to the DRAM. This is because the DRAM controller contains no logic to take care of DRAM initialization, programming the mode registers, or putting the DRAM into or out of the sleep and standby modes like self-refresh. Essentially, these functions are made available over the peripheral bus. To program the mode registers, the DRAM controller needs to be put in a bypass mode, where incoming requests are not serviced. In this bypass mode, commands are sent from the peripheral interface directly to the DRAM to program the mode registers or to put the DRAM into or out of sleep mode.
- During bypass mode, all reads and writes are blocked. Refresh keeps running, but can be separately disabled.

Chapter 14

DRAM Controller Priority Manager

14.1 Introduction

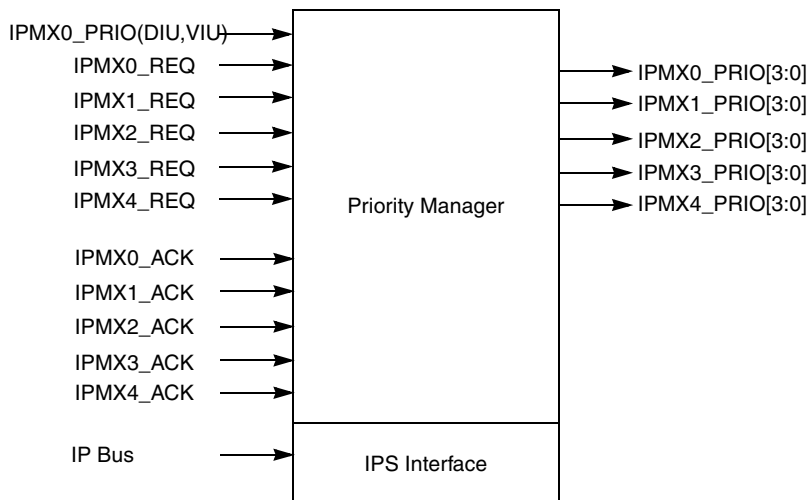


Figure 14-1. Priority Manager Block Diagram

The multi-port DRAM controller services the highest priority request from five different busses using a 4-bit priority signal. This 4-bit priority is dynamically set by the DRAM controller priority manager based on register settings and the most recent activity on each bus. In general, the DRAM priority manager increases the priority of a channel if it has not been recently serviced and decreases the priority of channels that have been recently serviced.

A block diagram of the priority manager is given in [Figure 14-1](#). It accepts the request and ACK-signals for all five DRAM busses, and produces the priority signals for the five busses.

The priority manager uses an ACK-based schema; the priority is dependent on how many times for the last N requests accepted by the DRAM controller the current own bus won the request. A running average counter keeps track of how many acknowledgements out of the last N acknowledgements have been for a specific bus. This number is then put in a look-up table configurable by writing some config registers. The output of the look-up table is the priority for the next request on this bus.

This priority schema is versatile because programming the look-up table allows controlling relative priority to other channels and the average share of the bandwidth the current master gets. The priority schema introduces fairness because the look-up table can be programmed to reduce the priority of a bus that has won a large share of requests and increase the priority of a bus that lost a large share of requests.

14.1.1 Features

- Dynamic priority calculation based on ACKing history
 - Fully programmable using look-up table.
 - Can be configured for high or low latency and high or low bandwidth
 - Separate control over average latency and average bandwidth
 - Versatile so it can mimic the CSB arbitration schema.
 - Fairness guaranteed by reducing priority of channels that receive a lot of grants, and increasing priority of channels that are denied the bus often.
 - Repeat transfer built into the DRAM controller. Priority manager can set the maximum repeat count by controlling when lowest priority occurs.
- Feed-through mode where DIU and VIU priority is controlled directly by the DIU and VIU.

14.2 Bus Connections

The following masters are connected to five busses.

- Bus 0 : DIU, VIU
- Bus 1 : Power architecture e300, PCI
- Bus 2 : AXE audio engine
- Bus 3 : MBX graphics engine¹
- Bus 4 : USB, DMA, FEC, SATA

14.3 Memory Map and Register Definition

14.3.1 Memory Map

Table 14-1. Prioman Memory Map

Offset or Address	Register	Access	Section/Page
0x80	PRIOMAN_CONFIG1	R/W	14.3.2.1/14-4
0x84	PRIOMAN_CONFIG2	R/W	14.3.2.1/14-4
0x88	HIPRIO_CONFIG	R/W	14.3.2.2/14-6
0x8C	LUT table 0 main upper	R/W	14.3.2.3/14-7
0x90	LUT table 1 main upper	R/W	14.3.2.3/14-7
0x94	LUT table 2 main upper	R/W	14.3.2.3/14-7
0x98	LUT table 3 main upper	R/W	14.3.2.3/14-7
0x9C	LUT table 4 main upper	R/W	14.3.2.3/14-7
0xA0	LUT table 0 main lower	R/W	14.3.2.4/14-8
0xA4	LUT table 1 main lower	R/W	14.3.2.4/14-8

1. Only on MPC5121e. Not on MPC5123

Table 14-1. Prioman Memory Map (continued)

Offset or Address	Register	Access	Section/Page
0xA8	LUT table 2 main lower	R/W	14.3.2.4/14-8
0xAC	LUT table 3 main lower	R/W	14.3.2.4/14-8
0xB0	LUT table 4 main lower	R/W	14.3.2.4/14-8
0xB4	LUT table 0 alternate upper	R/W	14.3.2.5/14-9
0xB8	LUT table 1 alternate upper	R/W	14.3.2.5/14-9
0xBC	LUT table 2 alternate upper	R/W	14.3.2.5/14-9
0xC0	LUT table 3 alternate upper	R/W	14.3.2.5/14-9
0xC4	LUT table 4 alternate upper	R/W	14.3.2.5/14-9
0xC8	LUT table 0 alternate lower	R/W	14.3.2.6/14-10
0xCC	LUT table 1 alternate lower	R/W	14.3.2.6/14-10
0xD0	LUT table 2 alternate lower	R/W	14.3.2.6/14-10
0xD4	LUT table 3 alternate lower	R/W	14.3.2.6/14-10
0xD8	LUT table 4 alternate lower	R/W	14.3.2.6/14-10
0xDC	Performance monitor config	R/W	14.3.2.7/14-11
0xE0	Event time counter	R/W	14.3.2.8/14-12
0xE4	Event time preset	R/W	14.3.2.9/14-13
0xE8	Performance monitor 1 address low	R/W	14.3.2.10/14-13
0xEC	Performance monitor 2 address low	R/W	14.3.2.10/14-13
0xF0	Performance monitor 1 address hi	R/W	14.3.2.10/14-13
0xF4	Performance monitor 2 address hi	R/W	14.3.2.10/14-13
0x100	Performance monitor 1 read counter	R	14.3.2.11/14-14
0x104	Performance monitor 2 read counter	R	14.3.2.11/14-14
0x108	Performance monitor 1 write counter	R	14.3.2.11/14-14
0x10C	Performance monitor 2 write counter	R	14.3.2.11/14-14
0x110	Granted ack counter 0	R	14.3.2.12/14-15
0x114	Granted ack counter 1	R	14.3.2.12/14-15
0x118	Granted ack counter 2	R	14.3.2.12/14-15
0x11C	Granted ack counter 3	R	14.3.2.12/14-15
0x120	Granted ack counter 4	R	14.3.2.12/14-15
0x124	Cumulative wait counter 0	R	14.3.2.13/14-15
0x128	Cumulative wait counter 1	R	14.3.2.13/14-15
0x12C	Cumulative wait counter 2	R	14.3.2.13/14-15
0x130	Cumulative wait counter 3	R	14.3.2.13/14-15
0x134	Cumulative wait counter 4	R	14.3.2.13/14-15
0x138	Summed priority counter 0	R	14.3.2.14/14-16
0x13C	Summed priority counter 1	R	14.3.2.14/14-16

Table 14-1. Prioman Memory Map (continued)

Offset or Address	Register	Access	Section/Page
0x140	Summed priority counter 2	R	14.3.2.14/14-16
0x144	Summed priority counter 3	R	14.3.2.14/14-16
0x148	Summed priority counter 4	R	14.3.2.14/14-16

14.3.2 Register Descriptions

14.3.2.1 PRIOMAN_CONFIG1, PRIOMAN_CONFIG2

0x80 PRIOMAN_CONFIG1																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R			LUT SEL4[1:0]		LUT SEL3[1:0]		LUT SEL2[1:0]		LUT SEL1[1:0]		LUT SEL0[1:0]		ACK_COUNT4[3:0]				
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	ACK_COUNT3[2:0]				ACK_COUNT2[3:0]				ACK_COUNT1[3:0]				ACK_COUNT0[3:0]				
W																	
Reset	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	


 = Unimplemented or Reserved

Figure 14-2. PRIOMAN_CONFIG1 Register

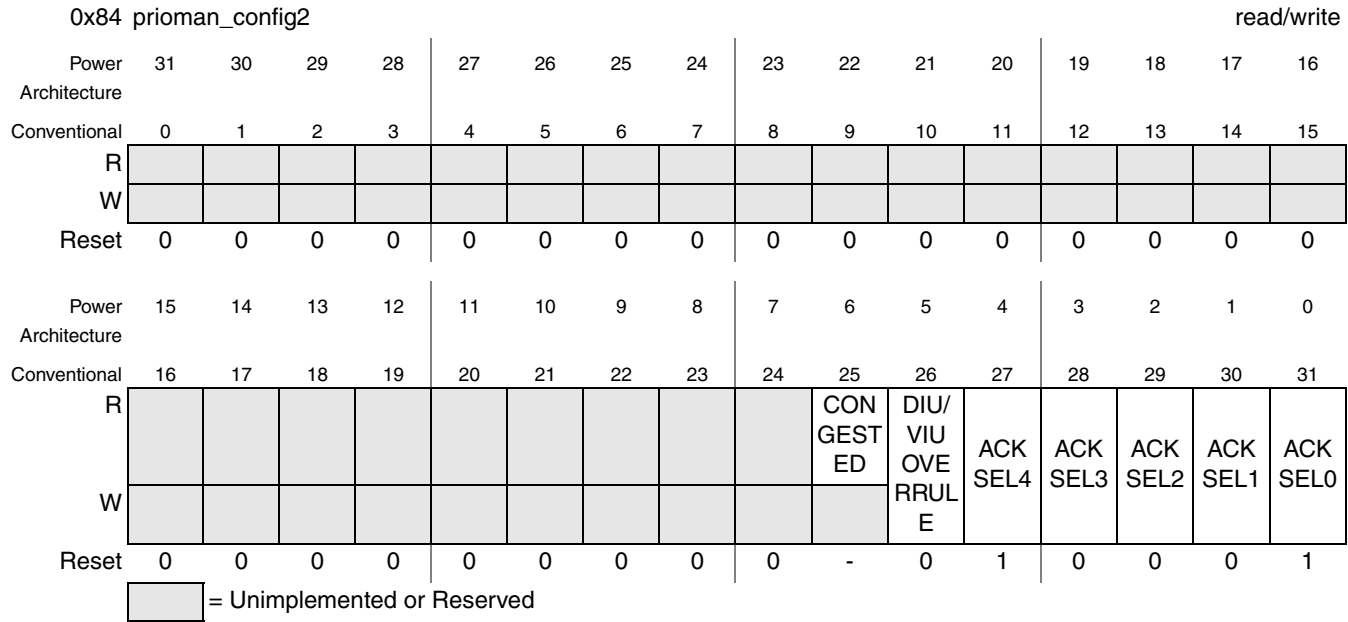


Figure 14-3. PRIOMAN_CONFIG2 Register

Table 14-2. PRIOMAN CONFIG Fields

Field	Description
CONGESTED	Read only 1: Congested flag is set 0: Congested flag is cleared
DIU/VIU-OVER RULE	1: Priority for channel 0 taken from DIU/VIU directly 0: DIU/VIU priority follows normal schema.
ACK SEL4 ACK SEL3 ACK SEL2 ACK SEL1 ACK SEL0	There is one of these bits for each priority manager channel. They determine what happens if the current channel is not requesting 1: If current channel is not requesting, every ACK for other channel is treated like an ACK for the current channel. Regulates default priority to low value 0: No special overrule. Regulates default priority to high value

Table 14-2. PRIOMAN CONFIG Fields

Field	Description
LUT_SEL4 LUT_SEL3 LUT_SEL2 LUT_SEL1 LUT_SEL0	Selectors between primary and secondary Look-Up table configuration register 0: Select main look-up table configuration register 1: Select alternate look-up table configuration register 2: Select alternate look-up table configuration register if congested flag is set ¹ 3: Select alternate look-up table configuration register if DIU/VIU incoming priority bit 3 is high. ²
ACK_COUNT4 [3:0] ACK_COUNT3 [3:0] ACK_COUNT2 [3:0] ACK_COUNT1 [3:0] ACK_COUNT0 [3:0]	Configuration fields. One for every channel. Determines how many requests the number of ACKs for the self-channel is counted. ³ 0 : 1 1 : 2 2 : 3 3 : 4 4 : 6 5 : 8 6 : 12 7 : 16 8 : 24 9 : 32 10 : 48 11 : 63

¹ Congested flag is explained in [Section 14.3.2.2, “HIPRIO_CONFIG”](#)

² The switch for all tables is based on the DIU/VIU flag. If LUT_SEL1 = 3, the e300/PCI table switches to the alternate table if the DIU/VIU flag is set.

³ Look-up table input is running average of the number of acks for the self channel counted over the grant total of the last N acks. ack_count[2:0] controls the parameter N.

14.3.2.2 HIPRIO_CONFIG

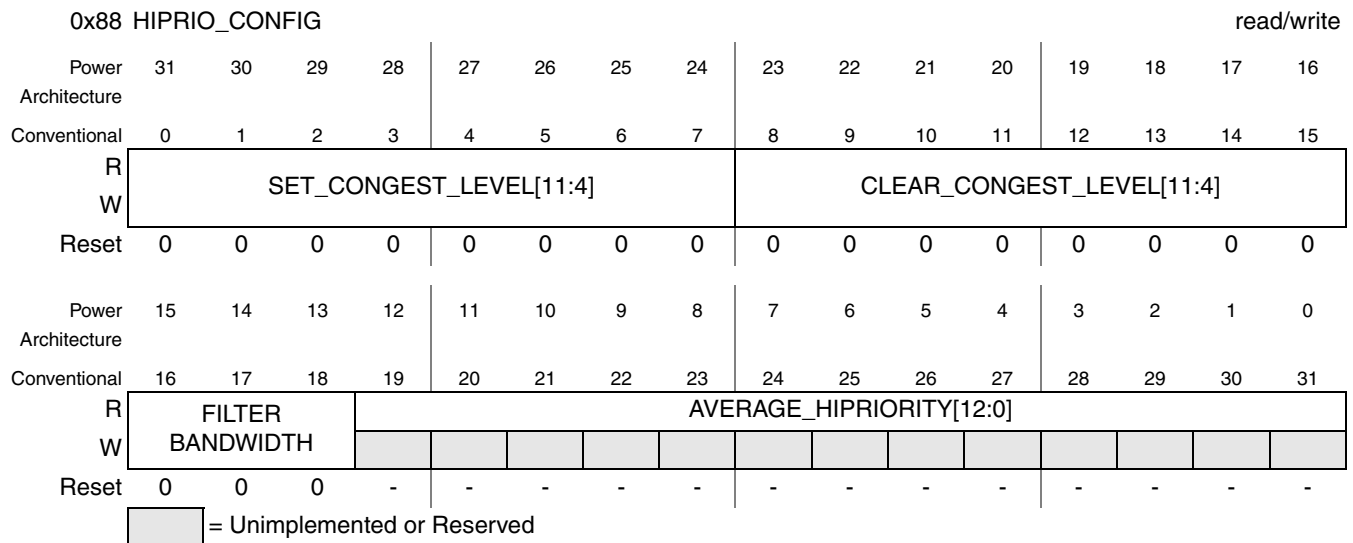


Figure 14-4. HIPRIO_CONFIG Register

The HIPRIO_CONFIG register controls the hiprio detection logic. The hiprio detection logic detects what percentage of the requests ACKed by the DRAM controller are ACKed with a priority larger than eight.

Table 14-3. HIPRIO_CONFIG Fields

Field	Description
AVERAGE_HIPRIORITY[12:0]	Average number of high priority requests to DRAM, coded between values 0x1000 and 0x0000 0x1000: 100% high-priority requests 0x0000: 0% high-priority requests
FILTER BANDWIDTH[2:0]	This setting controls the averaging time of the filter used for average_hipriority[12:0] ¹ 0: Time constant W0 = 8 ACKS, K = 0.125 1: Time constant W0 = 16 ACKS, K = 0.0625 2: Time constant W0 = 32 ACKS, K = 0.0312 3: Time constant W0 = 64 ACKS, K = 0.0156 4: Time constant W0 = 128 ACKS, K = 0.0078 5: Time constant W0 = 256 ACKS, K = 0.0039 6: Time constant W0 = 512 ACKS, K = 0.0020 7: Time constant W0 = 1024 ACKS, K = 0.0010
SET_CONGEST_LEVEL[11:4]	If(average_hipriority[12:4] > set_congest_level[12:4]) -> set the congested flag
CLEAR_CONGEST_LEVEL[112:4]	If(average_hipriority[12:4] < clear_congest_level[12:4]) -> clear the congested flag

¹ Refer to [Equation 14-1](#) and [Equation 14-2](#) for relationship between filter bandwidth and filter behavior.

14.3.2.3 LUT0 – LUT4 Main Upper

0x8C LUT0 main upper																read/write	
0x90 LUT1 main upper																	
0x94 LUT2 main upper																	
0x98 LUT3 main upper																	
0x9C LUT4 main upper																	
Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	PRIO15[3:0]				PRIO14[3:0]				PRIO13[3:0]				PRIO12[3:0]				
W																	
Reset	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	PRIO11[3:0]				PRIO10[3:0]				PRIO9[3:0]				PRIO8[3:0]				
W																	
Reset	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	

Figure 14-5. LUT0 – LUT4 Main Upper Register

These registers contain the upper eight entries of the look-up tables for channels 0 to 4, main table. All registers contain identical fields.

Table 14-4. LUT Table Main Upper Fields

Field	Description
PRI015[3:0]	Priority setting if 15 or more ACK's for own channel counted
PRI014[3:0]	Priority setting if 14 ACK's for own channel counted
PRI013[3:0]	Priority setting if 13 ACK's for own channel counted
PRI012[3:0]	Priority setting if 12 ACK's for own channel counted
PRI011[3:0]	Priority setting if 11 ACK's for own channel counted
PRI010[3:0]	Priority setting if 10 ACK's for own channel counted
PRI09[3:0]	Priority setting if 9 ACK's for own channel counted
PRI08[3:0]	Priority setting if 8 ACK's for own channel counted

14.3.2.4 LUT0 – LUT4 Main Lower

0xA0 LUT0 main lower

0xA4 LUT1 main lower

0xA8 LUT2 main lower

0xAC LUT3 main lower

0xB0 LUT4 main lower

read/write

Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRI07[3:0]				PRI06[3:0]				PRI05[3:0]				PRI04[3:0]			
W																
Reset	0	0	1	0	0	0	1	1	0	0	1	1	0	1	0	0
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRI03[3:0]				PRI02[3:0]				PRI01[3:0]				PRI00[3:0]			
W																
Reset	0	1	0	1	0	1	1	0	0	1	1	1	1	0	1	0

Figure 14-6. LUT – LUT4 Main Lower Register

These registers contain the upper eight entries of the look-up tables for channels 0 to 4, main table. All registers contain identical fields.

Table 14-5. LUT Table Main Lower Fields

Field	Description
PRI07[3:0]	Priority setting if seven ACKs for own channel counted
PRI06[3:0]	Priority setting if six ACKs for own channel counted
PRI05[3:0]	Priority setting if five ACKs for own channel counted
PRI04[3:0]	Priority setting if four ACKs for own channel counted
PRI03[3:0]	Priority setting if three ACKs for own channel counted

Table 14-5. LUT Table Main Lower Fields

Field	Description
PRI02[3:0]	Priority setting if two ACKs for own channel counted
PRI01[3:0]	Priority setting if one ACK for own channel counted
PRI00[3:0]	Priority setting if zero ACKs for own channel counted

14.3.2.5 LUT0 – LUT4 Alternate Upper

0xB4 LUT0 alternate upper

read/write

0xB8 LUT1 alternate upper

0xBC LUT2 alternate upper

0xC0 LUT3 alternate upper

0xC4 LUT4 alternate upper

Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRIO15[3:0]				PRIO14[3:0]				PRIO13[3:0]				PRIO12[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRIO11[3:0]				PRIO10[3:0]				PRIO9[3:0]				PRIO8[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-7. LUT – LUT4 Alternate Upper Register

These registers contain the upper eight entries of the look-up tables for channels 0 to 4, main table. All registers contain identical fields.

Table 14-6. LUT Table Alternate Upper Fields

Field	Description
PRI015[3:0]	Priority setting if 15 or more ACKs for own channel counted
PRI014[3:0]	Priority setting if 14 ACKs for own channel counted
PRI013[3:0]	Priority setting if 13 ACKs for own channel counted
PRI012[3:0]	Priority setting if 12 ACKs for own channel counted
PRI011[3:0]	Priority setting if 11 ACKs for own channel counted
PRI010[3:0]	Priority setting if 10 ACKs for own channel counted
PRI09[3:0]	Priority setting if 9 ACKs for own channel counted
PRI08[3:0]	Priority setting if 8 ACKs for own channel counted

14.3.2.6 LUT0 – LUT4 Alternate Lower

0xC8 LUT0 alternate lower																read/write	
0xCC LUT1 alternate lower																	
0xD0 LUT2 alternate lower																	
0xD4 LUT3 alternate lower																	
0xD8 LUT4 alternate lower																	
Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	PRIO7[3:0]				PRIO6[3:0]				PRIO5[3:0]				PRIO4[3:0]				
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	PRIO3[3:0]				PRIO2[3:0]				PRIO1[3:0]				PRIO0[3:0]				
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 14-8. LUT Table [4:0] Alternate Lower Register

These registers contain the upper eight entries of the look-up tables for channels 0 to 4, alternate table. All registers contain identical fields.

Table 14-7. LUT Table Alternate Lower Fields

Field	Description
PRIO7[3:0]	Priority setting if seven ACKs for own channel counted
PRIO6[3:0]	Priority setting if six ACKs for own channel counted
PRIO5[3:0]	Priority setting if five ACKs for own channel counted
PRIO4[3:0]	Priority setting if four ACKs for own channel counted
PRIO3[3:0]	Priority setting if three ACKs for own channel counted
PRIO2[3:0]	Priority setting if two ACKs for own channel counted
PRIO1[3:0]	Priority setting if one ACK for own channel counted
PRIO0[3:0]	Priority setting if zero ACKs for own channel counted

14.3.2.7 PERMON_CONFIG

0xDC permon_config																read/write	
Power	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Architecture																	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	INT			DMA REQ		EVEN TCO UNTFREE RUN											
W		INTCLEAR	INTEN		DMA REQ STOP		EVEN TCO UNTT RIG										
Reset	—	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Power	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Architecture																	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R																	
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-9. Performance Monitor Config Register

Table 14-8. Performance Monitor Config Register Fields

Field	Description
INT	Read-Only Sticky Bit. Interrupt pending register. Set when interrupt is made pending.
INTCLEAR	Write-Only. Writing this bit 1 clears bit INT to zero.
INTEN	Interrupt Enable. When this bit is 1 and bit INT is 1, the processor gets an interrupt request.
DMAREQ	Read-Only. DMA request. Set when event counter time reaches zero. Cleared when first counter register is read.
DMAREQSTOP ¹	Read/Write. When this bit is 1, the DMA request is cleared and cannot get set. When this bit is 0, the DMAreq functions as expected.
EVENTCOUNT FREERUN	1: Event Counter Free Run. After reaching zero, the event time counter is reloaded from event time preset and a new cycle starts. 0 : Event Counter Single-Shot. After reaching zero, the event time counter stays at 0 and is not reloaded.
EVENTCOUNT TRIGGER	Write-Only Bit. Writing to this bit causes all count registers to be transferred to the buffer registers, and subsequent be cleared. It causes the event counter to be reloaded from the event time preset register. No interrupt or DMA request is generated on writing this register, but both are generated when the event time counter register reaches zero.
LUT SEL4 LUT SEL3 LUT SEL2 LUT SEL1 LUT SEL0	Selectors between primary and secondary look-up table configuration register These selectors determine which LUT table is used for the summed priority counters. The priorities entered into these counters may depend on a different LUT table than the priorities sent to the DRAM controller. 0: Select main look-up table configuration register 1: Select alternate look-up table configuration register 2: Select alternate look-up table configuration register if congested flag is set ² 3: Select alternate look-up table configuration register if DIU/VIU incoming priority bit 3 is high

¹ This bit should be set as long as the DMA channel is not configured to manage the request. After configuring the DMA, clear the bit, and data starts to be transferred on every time tick.

² Congested flag is explained in section congestion detector.

14.3.2.8 Event Time Counter

0xE0 EVENT_TIME_COUNTER																read/write	
Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R									EVENT_TIME_COUNTER[23:0]								
W																	
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	EVENT_TIME_COUNTER[23:0]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 14-10. Event Time Counter Register

This 32-bit register has only one field - the 24-bit EVENT_TIME_COUNTER. The counter decrements to zero. The interrupt and the DMA request are made pending when it reaches zero. On reaching zero, the counter reloads from event count preset register if the bit eventCountFreeRun is set in the perfmon_config register.

On reaching zero, all performance monitor count registers are loaded in the performance monitor buffer registers, and cleared.

The register is read/write.

14.3.2.9 Event Time Preset

0xE4 EVENT_TIME_PRESET																read/write	
Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R									EVENT_TIME_PRESET[23:0]								
W																	
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	EVENT_TIME_PRESET[23:0]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 14-11. Event Time Preset Register

The EVENT_TIME_PRESET register contains the 24-bit preset value to be loaded into the event time counter register in case this preloads.

14.3.2.10 Performance Monitor 1 and 2 Address Registers

0xE8 Performance monitor 1 address low 0xEC Performance monitor 2 address low 0xF0 Performance monitor 1 address high 0xF4 Performance monitor 2 address high																read/write	
Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	PERFORMANCE MONITOR {1,2} ADDRESS {LOW,HIGH}[31:16]																
W																	
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	PERFORMANCE MONITOR {1,2} ADDRESS {LOW, HIGH} [15:5]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 14-12. Performance Monitor Address Registers

These registers determine if a Power Architecture (e300) processor access hits in the performance monitor 1 or performance monitor 2 address space.

- If ((e300 CPU address >= performance monitor 1 address low) && (e300 CPU address < performance monitor 1 address hi))
Increment *performance monitor 1 read counter* on reads
Increment *performance monitor 1 write counter* on writes.
- If ((e300 CPU address >= performance monitor 2 address low) && (e300 CPU address < performance monitor 3 address hi))
Increment *performance monitor 2 read counter* on reads
Increment *performance monitor 2 write counter* on writes.

14.3.2.11 Performance Monitor Counters

0x100 Performance monitor 1 read counter
 0x104 Performance monitor 2 read counter
 0x108 Performance monitor 1 write counter
 0x10C Performance monitor 2 write counter

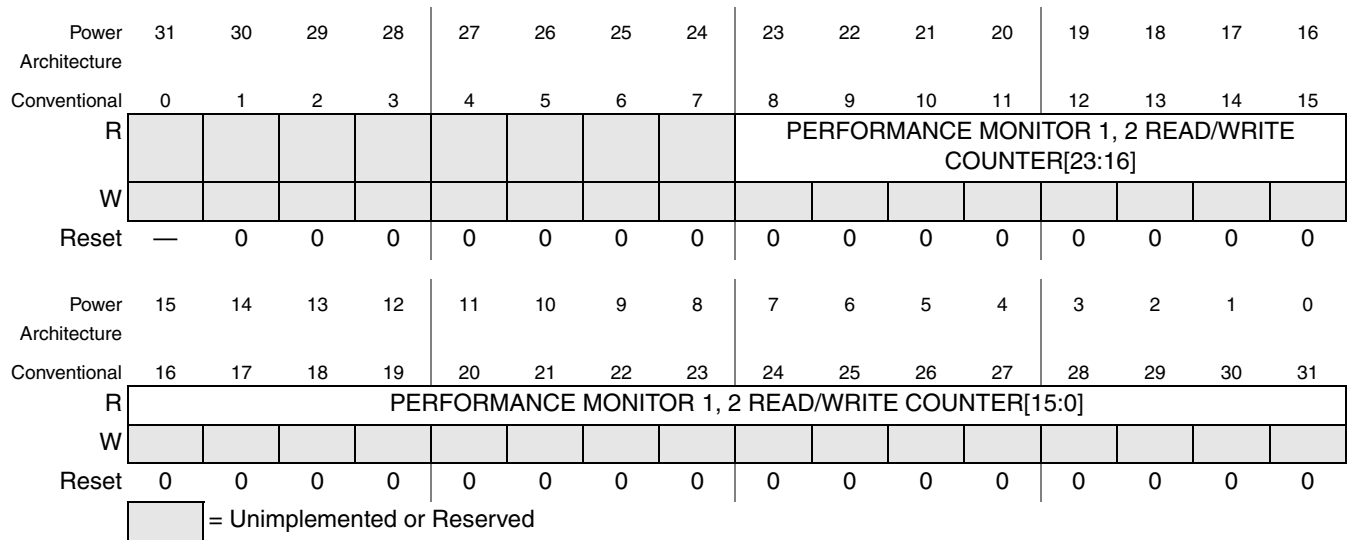


Figure 14-13. Performance Monitor Counter Registers

14.3.2.12 Granted Ack Counters

0x110 Granted ACK counter 0
 0x114 Granted ACK counter 1
 0x118 Granted ACK counter 2
 0x11C Granted ACK counter 3
 0x120 Granted ACK counter 4

read

Power	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Architecture																
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									GRANTED ACK COUNTER 0 – 4[23:16]							
W																
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Architecture																
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GRANTED ACK COUNTER 0 – 4[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-14. Granted ACK Counter 0 – 4 Registers

14.3.2.13 Cumulative Wait Counters

0x124 Cumulative wait counter 0
 0x128 Cumulative wait counter 1
 0x12C Cumulative wait counter 2
 0x130 Cumulative wait counter 3
 0x134 Cumulative wait counter 4

read

Power	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Architecture																
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									CUMULATIVE WAIT COUNTER 0 – 4[23:16]							
W																
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Architecture																
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CUMULATIVE WAIT COUNTER 0 – 4[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-15. Cumulative Wait Counter 0 – 4 Registers

14.3.2.14 Summed Priority Counters

0x138	Summed priority counter 0	read
0x13C	Summed priority counter 1	
0x140	Summed priority counter 2	
0x144	Summed priority counter 3	
0x148	Summed priority counter 4	

Power Architecture	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Conventional	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									SUMMED PRIORITY COUNTER 0 – 4[23:16]							
W																
Reset	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUMMED PRIORITY COUNTER 0 – 4[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-16. Summed Priority Counter 0 – 4 Registers

14.3.2.15 Counter Register Descriptions and Values

The counter registers contain 19 different 24-bit counter values. All these counter values count certain events. Table 14-9 gives the details on the nature of the event. Counter values Summed Priority Counter 2, Summed Priority Counter 3 and Summed Priority Counter 4 are available in two sets of registers. They are available in registers with the same name, but they are also available in a set of three other registers (granted ack counter or cumulative wait counter registers). The multiple-mapping of the three upper Summed Priority Counter registers allows easy and compact DMA transfer to memory. Because of the multiple mapping, all 19 count values can be transferred to memory with a 64-byte DMA transfer starting at address 0x100. The multiple mapping allows the DMA to get all information with a 64-byte transfer, but some decompression is needed on decoding the data, while the CPU can read the 19 registers and mask out the upper eight bits to get relevant information.

Table 14-9. Monitor Counter Descriptions

Field	Description
PERFORMANCE MONITOR 1-2 READ COUNTER	Every time the Processor or PCI performs a read access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor read counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.
PERFORMANCE MONITOR 1-2 WRITE COUNTER	Every time the Processor or PCI performs a write access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor write counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.
GRANTED ACK COUNTER 0-4	Every time the Multi-port DRAM controller grants a request for channel 0 – 4, the respective counter is incremented.
CUMULATIVE WAIT COUNTER 0-4	Every time there is a request pending to the multi-port DRAM controller for channel 0 – 4 and its not granted in the current cycle, the respective counter is incremented.
SUMMED PRIORITY COUNTER 0-4	Every time a request is granted by the multi-port DRAM controller for channel 0 – 4, a priority code is added to the respective counter. See text for details.

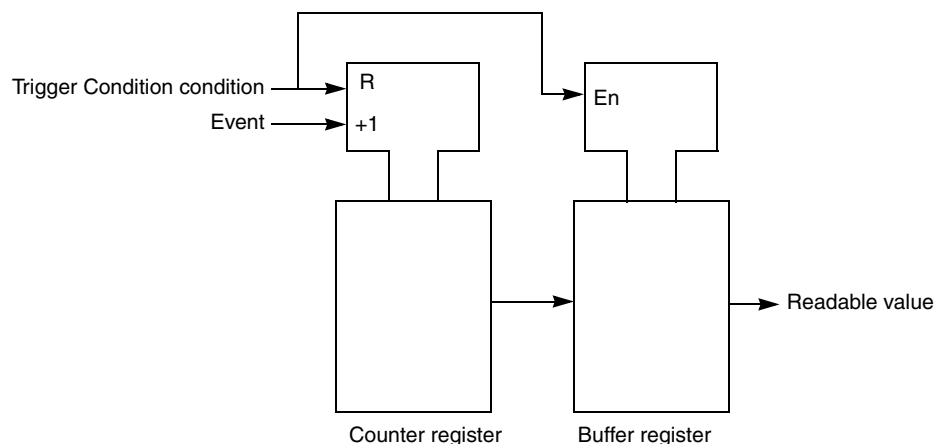


Figure 14-17. Monitor Counters

All counters in [Table 14-9](#) are double-buffered and [Figure 14-17](#) gives details. There are always two registers associated with every counter. The first register is the counter. It counts the events mentioned in the table. When the trigger condition occurs, the time event counter reaches zero, and the counter register is transferred to the buffer register, the counter register is then cleared. When accessing the register, the buffer register value is always returned.

The priority code added may or may not be the same as the priority code the request used on the DRAM controller. The codes are equal if the LUT SEL field for the channel is the same in registers PRIOMAN_CONFIG and PERFMON_CONFIG. If the LUT SEL fields differ, the field in PRIOMAN_CONFIG is used to calculate the channel priority code on the DRAM, and the field in PERFMON_CONFIG is used to calculate the priority code added to this register.

The possibility to use unequal LUT SEL fields makes it possible to use the main look-up tables for DRAM priority programming and the alternate look-up tables for performance monitoring. Making the look-up tables independent increases the possibility of what can be monitored.

14.4 Functional Description

The priority manager calculates the outgoing priority for all five channels of multi-port DRAM controller. The priority of any channel at a given time is a function of the request granting history of the DRAM controller. A granted request is called an ACK, so this schema is called an ACK-based schema, because the priority is determined by the history of which channels have been ack-ed in the past and when.

The priority manager calculates the priorities in a dynamic way. This means, a priority is never constant, but changes over time, even when the request is not serviced. As a request ages while its not being serviced, its priority escalates to a higher level , and as the level increases, it is eventually serviced.

The DRAM controller has a built-in preference to offer repeat for any incoming read request. The repeat goes on as long as the requesting channel keeps requesting, and its priority is greater than 0. When the outgoing priority for any channel is 0, the DRAM controller no longer services or repeats the request. This feature allows the priority manager to control the maximum repeat count for any incoming channel.

14.4.1 Description of Operation — Overview

Priority calculation for all channels is independent. There is no direct cross-dependency of the priority of one channel on the priority of another channel. The algorithm looks at the last N arbitration cycles on the bus. N is a programmable number, set by fields ack_count in register prio_man, described in [Figure 14-2](#). For the last N arbitration cycles, the number of times the own channel won the bus, is summed up, and saturated to a maximum of 15. This number of 0 to 15 is input into the applicable look-up table. LUT table 0 is for channel 0, LUT table 1 is for channel 1, and so on. The value for the particular number is the priority code going to the multiport DRAM controller. If N is set to 16 and the own channel was granted the bus four times in the last 16 bus grant, the index into the look-up table is four. The field prio4[3:0] of the relevant look-up table is the priority going to the multi-port DRAM controller.

There are two look-up tables for every channel, the main and the alternate. The algorithm may switch between both, depending on some settings. The default look-up table is the main. However, the alternate is used if:

- The particular channel has been configured to look at the DIU/VIU incoming priority, and the DIU/VIU incoming priority is eight or higher.
- The particular channel has been configured to look at the congestion monitor, and this block indicates the multi-port DRAM is congested.

14.4.2 Block Diagram

Figure 14-18 contains a block diagram of the block.

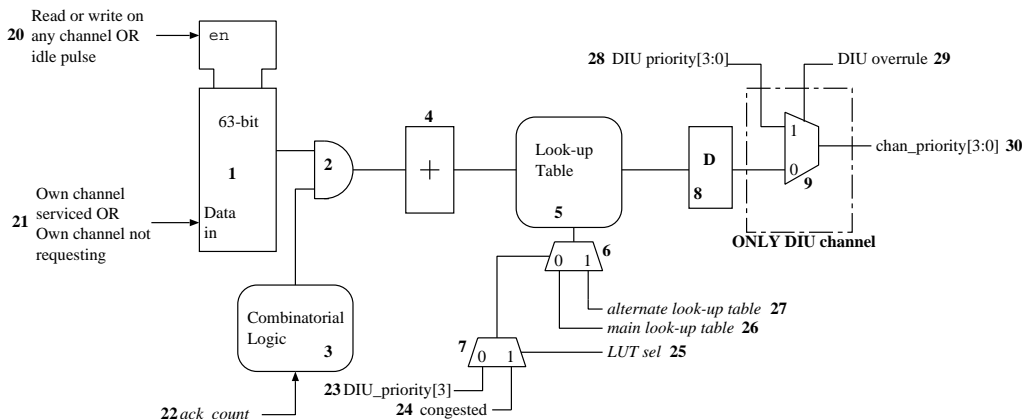


Figure 14-18. Priority Channel Block Diagram

Shift register shifts in information of the recent ACKs. Its a 63-stage shift register and contains information on the last 63 bus cycles of the DRAM controller.

The shift register is shifted any time a read or write request has been granted to the DRAM (An ACK to the requesting bus) or when there is an IDLE_PULSE. An idle pulse is generated every time the DRAM is idle for four consecutive clock cycles. Idle means none of the five incoming busses is making a request.

The shift data in is the corrected ACK for the self channel. If the shift register shifts because the current cycle is granted to the self channel, a 1 is shifted in, if not a 0 is shifted in. It always occurs like this when the own channel is requesting access. However, if the own channel is not requesting access, depending on control bit ACK_SEL, a 1 or a 0 is shifted in. If ACK_SEL is 1, a 1 is shifted in all the time when the self channel is not requesting and there is an ACK on any other channel or an idle pulse. If ACK_SEL is 0, zeros are shifted in.

The correction for the non-requesting channel allows you to steer the default priority, the priority that the channel gets, when it has not been requesting for some time. If ACK_SEL is set 1, the default priority is low. This setting is appropriate for peripherals with (large) FIFOs. When they are not requesting, the FIFO is quite full. When they do get on the bus, they can start with low priority and escalate to higher after some time.

Setting ACK_SEL to 0 is appropriate for peripherals that desire high priority. The Power Architecture processor and AXE core are in this case. When they are not on the bus, it's because they find the instruction or data that they need in the processor caches, so they don't request. When the cache misses, the request comes on the bus, and needs to be serviced fast. Therefore, ACK_SEL is set 0, the default priority is high and servicing fast. If Power Architecture Processor and/or AXE get on the bus a lot (due to a lot of cache swapping), the priority manager detects this and degrades their priorities over time. The other masters continue to receive their fair share of bus bandwidth.

The output of the shift register is ANDed in to look at only the last N ACKs. Combinatorial logic decodes the ANDing code from register field ACK_COUNT. The number of ones after the ANDing is added up in ADDER 4 and saturated. The result out of ADDER 4 is a number from 0 to 15. This number is input in the look-up table. Table look-up content is taken for channel 1 from register lut table 1 main[63:0] or lut table 1 alternate[63:0]. Because of the 64-bit nature of the registers, four 32-bit registers are involved. The description is given in [Figure 14-5](#), [Figure 14-6](#), [Figure 14-7](#), and [Figure 14-8](#).

The MUX selects whether to use the main or the alternate register. The MUX condition has two possible sources again, selected by MUX 7, by means of control bit LUT SEL described in register PRIOMAN_CONFIG, with details in [Figure 14-2](#).

If LUT SEL is 1, the alternate table is selected when the multi-port controller is congested. If LUT sel is 0, the alternate table is selected when DIU/VIU incoming priority is higher than eight.

Pipeline register is present purely for implementation reasons. It has no algorithmic function.

For the DIU and VIU, an additional bypass mux is present. It overrides the prioman logic and inserts the incoming DIU/VIU priority in the output if control bit DIU/VIU overrule is set. This bit is present in register PRIOMAN_CONFIG, with details in [Figure 14-2](#).

14.4.3 Congestion Detector

The congestion detectors purpose is to detect when the multi-port DRAM controller is congested. Congestion is assumed if the share of the requests with priorities equal or greater than eight is more than a certain percentage. If congestion occurs, the priority manager may react by exchanging the look-up tables with the alternate look-up tables. This reduces the average priority of the incoming requests. The reduced priorities mean that on average, every incoming channel gets a lower priority and the DRAM controller tries harder to optimize on bandwidth and less to optimize to service the high-priority requests first. The switch-over is driven by the congestion state. If many requests come in on high priority, they all need to be serviced first, the congested flag goes high, and the controller reacts to this by reducing the request priorities (by switching in the alternate tables). Therefore, it can concentrate on the ones that are important and have room again for optimized bandwidth.

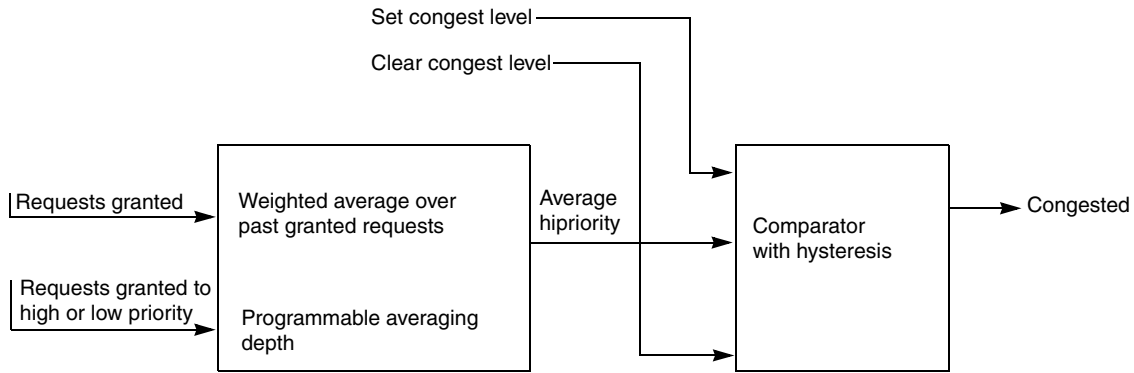


Figure 14-19. Congestion Detector – Simplified Block Diagram

A block diagram of the congestion detector is given in [Figure 14-19](#). The congestion detector consists of an averaging block, followed by a comparator with hysteresis. The averaging block calculates the weighted average of the percentage of high-priority requests, like given below.

$$\text{average priority} = \sum \text{weigh}(k) \cdot \text{val}(k) \quad \text{Eqn. 14-1}$$

The weighted average uses an exponential weighting when looking at the past granted requests. Requests granted in a more distant past have a lower weighting coefficient. The weighting coefficient uses an exponential back-off, following the formulae.

$$\text{weight}(k) = \frac{1}{W_0} \cdot \exp\left(-\frac{k}{W_0}\right) \quad \text{Eqn. 14-2}$$

In this formula, $\text{weight}(k)$ is the weighting coefficient used for the request granted k acknowledges ago, meaning k other requests have been granted after this one. The coefficient W_0 is programmable, dependent on the control field filter bandwidth in register `HIPRIO_CONFIG`, detailed in [Figure 14-4](#).

The value input in the weighting block, $\text{val}(k)$ is dependent on the priority of the request granted. It is 0 if the priority was seven or lower; it is 0x1000 if the priority was eight or higher.

The result of the weighted average is a number between 0 and 0x1000 input in the comparator with hysteresis. This result, `AVERAGE_HIPRIORITY`, can be monitored in register `HIPRIO_CONFIG` ([Figure 14-4](#)).

The weighted averaging block is followed by a comparator with hysteresis, with a programmable low threshold.

- If `AVERAGE_HIPRIORITY` is greater than `SET_CONGEST_LEVEL`, the congested flag is set.
- If `AVERAGE_HIPRIORITY` is lower than `CLEAR_CONGEST_LEVEL`, the congested flag is cleared.

Chapter 15

External Memory Bus (EMB)

15.1 Introduction

15.1.1 Overview

The EMB includes three different parallel interfaces. They are LocalPlus bus, NAND flash bus, and parallel ATA bus. The different buses are time multiplexed. The NFC bus and the ATA bus can work together at the same time. No multiplexing for this function is needed.

An EMB arbiter controls the multiplexing of the external pin (address and data lines) and grants the different bus masters to allow them to drive the external bus. The arbiter can be configured via the [EMB Share and Wait Count Register](#) and [EMB Pause Control Register](#) within the LPC memory map.

15.1.2 Features

- Arbitration between LPC and NFC/pATA
 - LPC CSB transfers cannot be paused.
 - LPC CSB request pauses NFC and pATA DMA transactions immediately or after share counter expires.
 - LPC FIFO request pauses NFC and pATA DMA transactions after share counter expires
 - pATA PIO transactions cannot be paused
 - pATA PIO request pauses LPC FIFO transaction within a BPT transfer (dynamic bus sizing), depending on the ATA_P bit setting ([EMB Pause Control Register](#))
 - pATA DMA and NFC requests cannot pause LPC FIFO transaction within a BPT transfer.
- Pin muxing between LPC and NFC/pATA

15.2 Functional Description

15.2.1 EMB Mux

The EMB mux switches, depending on EMB arbiter state, between the three different functions. The activated, granted module can drive the external bus.

[Table 15-1](#) describes which functionality is at the EMB bus depending on the activated, granted module.

Table 15-1. EMB_AD Multiplexing

Activated, Granted Module	Multiplexed Functionality at EMB_AD[31:16]	Multiplexed Functionality at EMB_AD[15:0]	Multiplexed Functionality at EMB_AX[2:0]
LPC	LPC_AD[31:16]	LPC_AD[15:0]	LPC_AX[2:0]
NFC/pATA	NFC_AD[15:0]	PATA_DATA[15:0]	PATA_ADDRESS[2:0]

Chapter 16

Fast Ethernet Controller (FEC)

16.1 Introduction

16.1.1 FEC Top Level

The block diagram of the FEC is shown in [Figure 16-1](#). To implement the FEC, a combination of hardware and microcode is employed. The network interfaces are shown on the bottom of the diagram, complying with industry and IEEE 802.3 standards.

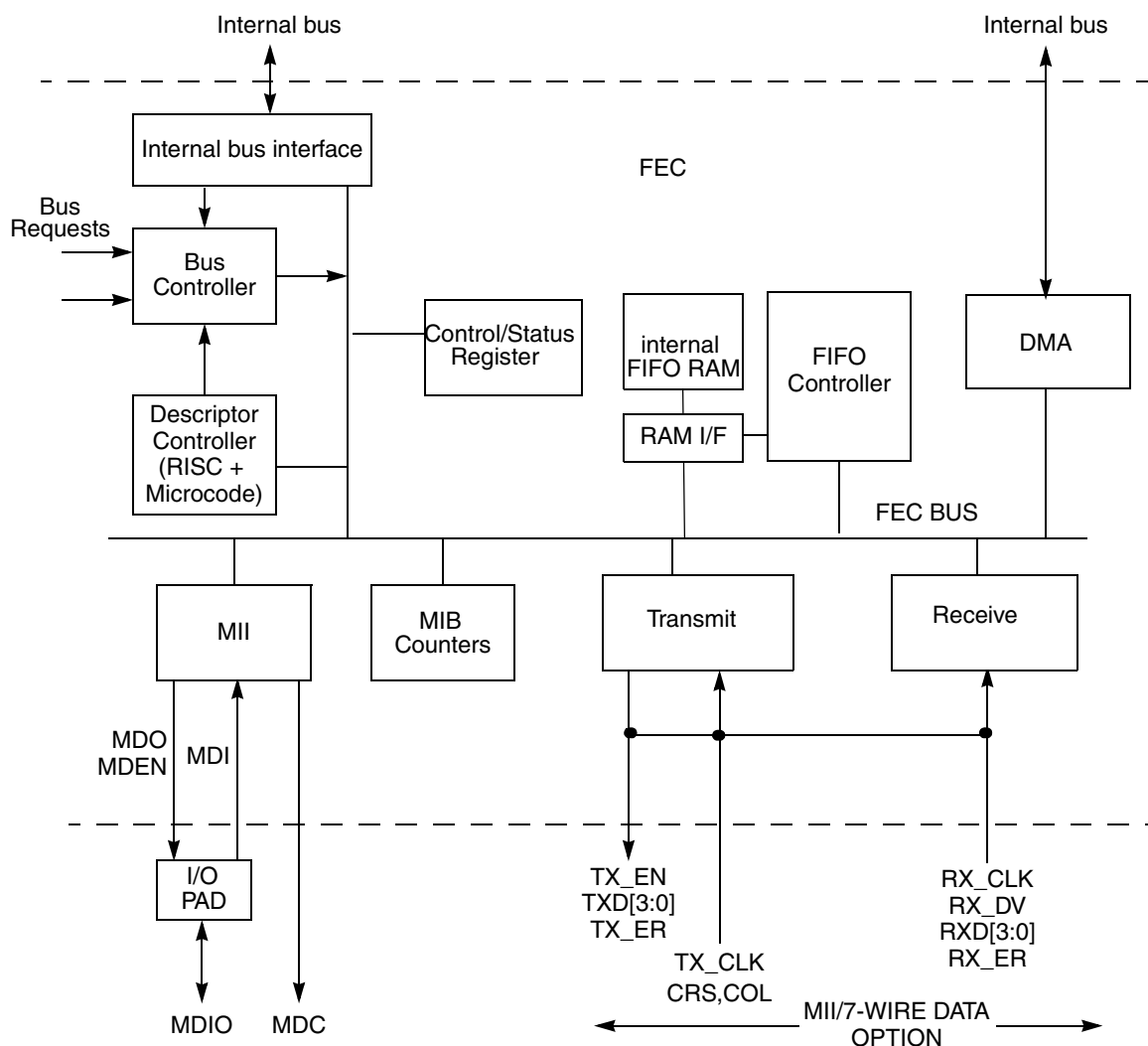


Figure 16-1. FEC Block Diagram

A RISC-based controller, called the descriptor controller, provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High-level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backfill timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine transfers FEC data only and is not related to the DMA controller in MPC5121e.

NOTE

The FIFO is used by FEC itself and can only be accessed by the DMA. You can configure the transmit/receive FIFO boundary(R_FSTART register).

The RAM is the central point of all data flow in the Fast Ethernet controller. The RAM is divided into transmit and receive FIFOs and the boundary is programmable (R_FSTART register). User data flows to/from the DMA unit from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

The bus controller decides which block is the tbus master on each clock. All of the blocks receive their control information from the tbus and, for the most part, provide status information over this same bus.

The user controls FEC by writing into control registers located in each block. The CSR(control and status register) block provides global control (e.g., Ethernet reset and enable, mode control) and interrupt managing registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the MDC (clock) and MDIO (bidirectional data) lines of the MII interface.

The FEC DMA block (not to be confused with DMA controller) provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode). Internal to these blocks are clock domain boundaries between the system clock and the network clocks.

The MIB block is an optional block that maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters.

16.1.2 Features

The fast Ethernet controller (FEC) incorporates several features/design goals important to its market:

- Support for different Ethernet physical interfaces:
 - 100 Mbps IEEE 802.3 MII
 - 10 Mbps IEEE 802.3 MII
 - 10 Mbps 7-wire interface (industry standard)
- IEEE 802.3 full-duplex flow control
- Programmable maximum frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbps throughput) with a minimum system clock rate of 50 MHz
- Support for half-duplex operation (100 Mbps throughput) with a minimum system clock rate of 25 MHz
- Retransmission from transmit FIFO following a collision (no processor bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)
 - Address recognition
 - Frames with broadcast address may always be accepted or always be rejected
 - Exact match for single 48-bit individual (unicast) address
 - Hash (64-bit hash) check of individual (unicast) addresses
 - Hash (64-bit hash) check of group (multicast) addresses
 - Promiscuous mode

16.1.3 Modes of Operation

The primary operational modes are described in this section.

- Full- and half-duplex operation

This is determined by the FDEN bit in the X_CNTRL register. Full-duplex mode is intended for use on point-to-point links between switches or end node to switch. Half-duplex mode is used in connections between an end node and a repeater or between repeaters.

Full-duplex flow control is an option that may be enabled in full-duplex mode. Refer to the RFC_PAUSE and TFC_PAUSE bits in [Section 16.3.5.12, “Transmit Control Register \(X_CNTRL\)”](#), the FCE bit in [Section 16.3.5.10, “Receive Control Register \(R_CNTRL\)”](#) and [Section 16.6.4, “Full-Duplex Flow Control”](#) for more details.
- 10 Mbps and 100 Mbps MII interface operation

The MAC-PHY interface operates in MII mode by asserting the MII_MODE bit in the R_CNTRL register. The MII is the media-independent interface defined by the 802.3 standard for 10/100 Mbps operation.

The speed of operation is determined by the TX_CLK and RX_CLK pins, which are driven by the transceiver. The transceiver auto-negotiates the speed or may be controlled by software via the

serial management interface (MDC/MDIO pins) to the transceiver. Refer to the MII_DATA and MII_SPEED register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.

- 10 Mbps 7-wire interface operation

The FEC support 7-wire interface used by many 10Mbps Ethernet transceivers. The MII_MODE bit in the R_CNTRL register controls this functionality. If this bit is cleared, MII mode is disabled and the 10Mbps 7-wire mode is enabled.

- Address recognition options

Refer to the R_CNTRL register for address recognition options. Also, refer to [Section 16.6.3, “Ethernet Address Recognition](#) for a detailed description. The options supported are promiscuous, broadcast reject, individual address hash, or exact match and multicast hash match.

- Internal loopback

Internal loopback mode is selected via the LOOP bit in the R_CNTRL register. Also, refer to [Section 16.6.7, “Internal and External Loopback](#) for a detailed description.

16.2 External Signal Description (Off Chip)

16.2.1 I/O Signal Overview

This section defines the FEC to chip pin I/O.

The FEC network interface supports multiple options. One is the MII option, which requires 18 I/O pins and supports both data and an out-of-band serial management interface to the PHY (transceiver) device. The MII option supports both 10 and 100 Mbps Ethernet rates. The second is referred to as the 7-wire interface and supports only 10 Mbps Ethernet data. The 7-wire interface uses a subset of the MII signals.

[Table 16-1](#) details the network interface signals. This table lists 18 signals, all of which are used for the 10/100 MII interface. The MDIO pin is bidirectional and corresponds to the MDI, MDO and MDIO pins on the FEC block. A subset of these signals is used for the 7-wire or SMII interface option.

Table 16-1. Signal Properties

Signal Name	Function	Reset
COL	MII - collision input 7-wire — collision input	undefined
CRS	MII — carrier sense input	undefined
MDC	MII — management clock output	0
MDIO	MII — management data bidirect	Hi-Z (input)
RX_CLK	MII — receive clock input 7 Wire — receive clock input	undefined
RX_DV	MII — receive data valid input 7-wire — rena input	undefined

Table 16-1. Signal Properties (continued)

Signal Name	Function	Reset
RDATA[3]	MII — receive data bit 3 input	undefined
RDATA[2]	MII — receive data bit 2 input	undefined
RDATA[1]	MII — receive data bit 1 input	undefined
RDATA[0]	MII — receive data bit 0 input 7-wire — receive data input	undefined
RX_ER	MII — receive error input	undefined
TX_CLK	MII — transmit clock input 7-wire — transmit clock input	undefined
TDATA[3]	MII — transmit data bit 3 output	undefined
TDATA[2]	MII — transmit data bit 2 output	undefined
TDATA[1]	MII — transmit data bit 1 output	undefined
TDATA[0]	MII — transmit data bit 0 output 7-Wire — transmit data output	undefined
TX_EN	MII — transmit data valid output 7-Wire — transmit data valid output	0
TX_ER	MII — transmit error output	0

16.2.2 Detailed Signal Descriptions

This section gives a detailed description of the Ethernet MAC-PHY Interface. First, an overview of Ethernet interfaces is presented, followed by a description of the interface signals. Next, the two different types of MII frames are described. Then, a brief overview of the MII management function is given. This is followed by a section on MII signal timing. Finally, the electrical specifications for this interface are given.

16.2.2.1 Ethernet MAC-PHY Interface

FEC support two kinds of Ethernet MAC-PHY interface: 7-wire and MII. A description of their interface follows.

16.2.2.1.1 Seven-Wire Ethernet MAC-PHY Interface

The Ethernet module can operate in a 10 Mbps mode using a 7-wire interface to an external physical interface. Serial mode connections to the external transceiver are defined in [Table 16-2](#).

Table 16-2. 7-Wire Interface

Signal Description	FEC Pin
Transmit clock	TX_CLK
Transmit enable	TX_EN
Transmit Data	TXD[0]
Collision	COL

Table 16-2. 7-Wire Interface (continued)

Signal Description	FEC Pin
Receive Clock	RX_CLK
Receive Enable	RX_DV
Receive Data	RXD[0]
Unused FEC inputs — tie to GND	RX_ER, CRS, RXD[3:1], MDI
Unused FEC outputs — ignore	TX_ER, TXD[3:1], MDC, MDO, MDEN

16.2.2.1.2 MII Ethernet MAC-PHY Interface

MI I interface is defined in the IEEE 803.3 standard. [Table 16-1](#) lists the MII interface with keyword MII.

16.2.2.2 Signal Description

The MII interface consists of 18 signals. The transmit and receive functions require seven signals each: four data signals, a delimiter, error, and clock. In addition, there are two signals that indicate the status of the media; one indicates the presence of a carrier and the second indicates a collision has occurred. The remaining two signals provide a management interface. Each MII signal is described in [Table 16-3](#).

Table 16-3. Detailed Signal Descriptions

Signal	I/O	Description	
TX_CLK	I	State Meaning	Asserted—A continuous clock that provides a timing reference for TX_EN, TXD, and TX_ER.
		Timing	Asserted—The frequency of TX_CLK is 25% of the transmit data rate, +/- 100 ppm. Duty cycle is 35%-65%, inclusive.
RX_CLK	I	State Meaning	Asserted—A continuous clock that provides a timing reference for RX_DV, RXD, and RX_ER.
		Timing	Asserted—The frequency of RX_CLK is 25% of the receive data rate, with a duty cycle between 35% and 65%.
TX_EN	O	State Meaning	Asserted—Assertion of this signal indicates there are valid nibbles being presented on the MII.
		Timing	Asserted—This signal is asserted with the first nibble of preamble and is negated prior to the first TX_CLK following the final nibble of the frame.
TXD	O	State Meaning	Asserted—TXD<3:0> represent a nibble of data when TX_EN is asserted and has no meaning when TX_EN is deasserted.
		Timing	Asserted— Table 16-1 summarizes the permissible encoding of TXD.
TX_ER	O	State Meaning	Asserted—Assertion of this signal for one or more clock cycles while TX_EN is asserted causes the PHY to transmit one or more illegal symbols.
		Timing	Asserted—Asserting TX_ER has no effect when operating at 10 Mbps or when TX_EN is deasserted. This signal transitions synchronously with respect to TX_CLK.
RX_DV	I	State Meaning	Asserted—When this signal is asserted, the PHY indicates a valid nibble is present on the MII.
		Timing	Asserted—This signal remains asserted from the first recovered nibble of the frame through the last nibble. Assertion of RX_DV must start no later than the SFD and excludes any EOF.

Table 16-3. Detailed Signal Descriptions (continued)

Signal	I/O	Description	
RXD	I	State Meaning	Asserted—RXD<3:0> represents a nibble of data to be transferred from the PHY to the MAC when RX_DV is asserted. A completely formed SFD must be passed across the MII.
		Timing	Asserted—When RX_DV is not asserted, RXD has no meaning. There is an exception to this explained later. Table 16-5 summarizes the permissible encoding of RXD.
RX_ER	I	State Meaning	Asserted—When RX_ER and RX_DV are asserted, the PHY has detected an error in the current frame.
		Timing	Asserted—When RX_DV is not asserted, RX_ER has no effect. This signal transitions synchronously with RX_CLK.
CRS	I	State Meaning	Asserted—This signal is asserted when the transmit or receive medium is not idle. If a collision occurs, CRS remains asserted through the duration of the collision.
		Timing	Asserted—This signal is not required to transition synchronously with TX_CLK or RX_CLK.
COL	I	State Meaning	Asserted—This signal is asserted upon detection of a collision and remains asserted while the collision persists. The behavior of this signal is not specified when in full-duplex mode.
		Timing	Asserted—This signal is not required to transition synchronously with TX_CLK or RX_CLK.
MDC	O	State Meaning	Asserted—This signal provides a timing reference to the PHY for data transfers on the MDIO signal. MDC is aperiodic and has no maximum high or low times.
		Timing	Asserted—The minimum high and low times are 160ns; the minimum period is 400ns.
MDIO	I/O	State Meaning	Asserted—This signal transfers control/status information between the PHY and MAC. It transitions synchronously to MDC. The MDIO pin is a bidirectional pin. The internal FEC signals that connect to this pad are MDI (data in), MDO (data out), and MD_EN (direction control, high for output).

[Table 16-4](#) below provides the interpretation of the possible encodings of TX_EN and TX_ER.

Table 16-4. MII: Valid Encoding of TXD, TX_EN and TX_ER

TX_EN	TX_ER	TXD	Indication
0	0	0000 through 1111	Normal inter-frame
0	1	0000 through 1111	Reserved
1	0	0000 through 1111	Normal data transmission
1	1	0000 through 1111	Transmit error propagation

A false carrier condition occurs if the PHY detects a bad start-of-stream delimiter. This condition is signaled to the MII by asserting RX_ER and placing 1110 on RXD. RX_DV must also be deasserted. The valid encodings of RX_DV, RX_ER, and RXD[3:0] are shown in [Table 16-5](#).

Table 16-5. MII: Valid Encoding of RXD, RX_ER, and RX_DV

RX_DV	RX_ER	RXD	Indication
0	0	0000 through 1111	Normal inter-frame
0	1	0000	Normal inter-frame
0	1	0001 through 1101	Reserved
0	1	1110	False carrier

Table 16-5. MII: Valid Encoding of RXD, RX_ER, and RX_DV (continued)

RX_DV	RX_ER	RXD	Indication
0	1	1111	Reserved
1	0	0000 through 1111	Normal data reception
1	1	0000 through 1111	Data reception with errors

16.2.2.2.1 MII Data Frame

Ethernet/802.3 data frames transmitted across the MII have the following format:

<inter-frame><preamble><sfd><data><efd>

The inter-frame period is an unspecified amount of time during which no data activity occurs on the MII. The deassertion of RX_DV and TX_EN indicates the absence of data activity.

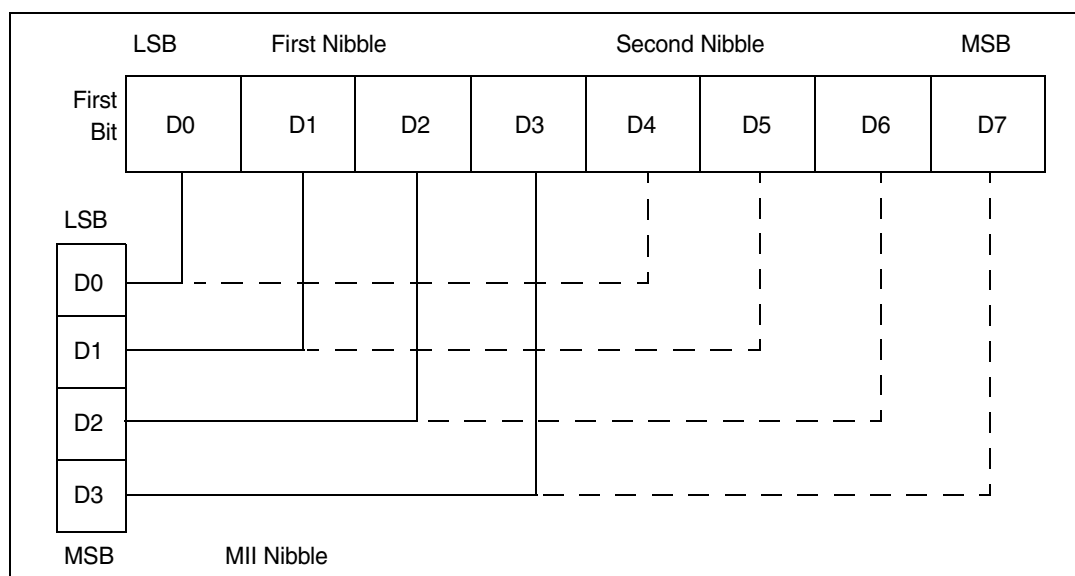
The preamble begins a frame and has a bit value of the following:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

The left-most 1 represents the LSB of the byte.

The SFD represents the start of a frame and has the bit value 10101011.

The data portion of the frame consists of N octets that corresponds to 2N nibbles being transmitted. The order of each nibble is defined in [Figure 16-2](#).

**Figure 16-2. MII Nibble/Octet to Octet/Nibble Mapping**

The end-of-frame delimiter is indicated by the deassertion of the TX_EN signal for data on TXD. For data on RXD, the deassertion of RX_DV constitutes an end-of-frame delimiter.

16.2.2.2 MII Management Frame Structure

A transceiver management frame transmitted on the MII management interface uses the MDIO and MDC pins. A transaction or frame on this serial interface has the following format:

<preamble><st><op><phyad><regad><ta><data><idle>

The (optional) preamble consists of a sequence of 32 continuous logic ones.

The start of frame (ST) is indicated by a <01> pattern.

The operation code (OP) for a read instruction is <10>. For a write operation, the operation code is <01>.

The PHYAD is a 5-bit field that allows for up to 32 PHYs to be addressed. The first address bit transmitted is the MSB of the address.

The REGAD is a 5-bit field that allows for 32 registers to be addressed within each PHY. The first register bit transmitted is the MSB of the address.

The TA field is a 2-bit field that provides spacing between the register address field and the data field, avoiding contention on the MDIO signal during a read operation.

The data field is 16 bits wide. The first data bit transmitted and received is data bit 15.

During the idle condition, MDIO is in the high impedance state.

The MII management register set located in the PHY may consist of a basic register set and an extended register set, as defined in [Table 16-6](#).

Table 16-6. MII Management Register Set

Register Address	Register Name	Basic/Extended
0	Control	B
1	Status	B
2,3	PHY Identifier	E
4	Auto-negotiation advertisement	E
5	AN link partner ability	E
6	AN expansion	E
7	AN next page transmit	E
8-15	Reserved	E
16-31	Vendor specific	E

16.3 Memory Map and Register Definition

16.3.1 Overview

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control, interrupts, and to extract status information. The descriptors are used to pass data buffers and related buffer or frame information between the hardware and software.

All accesses to and from the registers must be via 32-bit accesses. There is no support for accesses other than 32-bit.

This section defines the memory map and the registers, and then defines the buffer descriptors.

16.3.2 Top-Level Module Memory Map

The FEC implementation requires a 1-Kbyte memory map space. This is divided into two sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. [Table 16-7](#) defines the top-level memory map.

Table 16-7. Module Memory Map

Address	Function
0x000–1FF	Control/status registers
0x200–3FF	MIB block counters

16.3.3 Detailed Memory Map – Control/Status Registers

[Table 16-8](#) shows the address of the register, what block the register pertains to, the name of the register, and a brief description of the register.

The following table defines the subset of FEC address space used for control/status registers. These fall in the 000–1FF address offset range.

The block column indicates in which internal module the register resides.

Table 16-8. Control/Status Registers (Offset 000–1FF)

Offset or Address	Register	Mnemonic	Section/Page
000	FEC_ID register	FEC_ID	16.3.5.1/16-14
004	Interrupt event register	IEVENT	16.3.5.2/16-15
008	Interrupt mask register	IMASK	16.3.5.3/16-17
010	Receive ring updated flag	R_DES_ACTIVE	16.3.5.4/16-19
014	Transmit ring updated flag	X_DES_ACTIVE	16.3.5.5/16-20
024	Ethernet control register	ECNTRL	16.3.5.6/16-21
040	MII data register	MII_DATA	16.3.5.7/16-22
044	MII speed register	MII_SPEED	16.3.5.8/16-24

Table 16-8. Control/Status Registers (Offset 000–1FF) (continued)

Offset or Address	Register	Mnemonic	Section/Page
064	MIB control/status register	MIB_CONTROL	16.3.5.9/16-25
084	Receive control register	R_CNTRL	16.3.5.10/16-26
088	Receive hash	R_HASH	16.3.5.11/16-28
0C4	Transmit control register	X_CNTRL	16.3.5.12/16-29
0E4	Physical address low	PADDR1	16.3.5.13/16-31
0E8	Physical address high + type field	PADDR2	16.3.5.14/16-32
0EC	Opcode + pause duration	OP_PAUSE	16.3.5.15/16-33
118	Upper 32 bits of individual hash table	IADDR1	16.3.5.16/16-34
11C	Lower 32 bits of individual hash table	IADDR2	16.3.5.17/16-35
120	Upper 32 bits of group hash table	GADDR1	16.3.5.18/16-36
124	Lower 32 bits of group hash table	GADDR2	16.3.5.19/16-37
144	Transmit FIFO watermark	X_WMRK	16.3.5.20/16-38
14C	End of RAM	R_BOUND	16.3.5.21/16-39
150	Receive FIFO start address	R_FSTART	16.3.5.22/16-40
180	Beginning of receive descriptor ring	R_DES_START	16.3.5.23/16-41
184	Pointer to beginning of transmit descriptor ring	X_DES_START	16.3.5.24/16-42
188	Receive buffer size	R_BUFF_SIZE	16.3.5.25/16-43
1F4	DMA control for IP bus AMBA IF + DMA revision	DMA_CONTROL	16.3.5.26/16-44

16.3.4 MIB Block Counters Memory Map

Table 16-9 defines the MIB counters memory map, which defines the locations in the MIB RAM space where hardware-maintained counters reside. These fall in the 200–3FF address offset range. The counters are divided into two groups.

RMON counters cover the Ethernet statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet statistics group, a counter is included to count truncated frames because the FEC only supports frame lengths up to 2047 bytes. The RMON counters are implemented independently for transmit and receive to ensure accurate network statistics when operating in full-duplex mode.

IEEE counters support the mandatory and recommended counter packages defined in section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE basic package objects are supported by the FEC, but do not require counters in the MIB block. In addition, some of the recommended package objects that are supported do not require MIB counters. Counters for transmit and receive full-duplex flow control frames are also included.

Table 16-9. MIB Counters (Offset 200–3FF) (Sheet 1 of 3)

Address	Mnemonic	Description
0x200	RMON_T_DROP	Frames counted incorrectly
0x204	RMON_T_PACKETS	RMON TX packet count
0x208	RMON_T_BC_PKT	RMON TX broadcast packets
0x20C	RMON_T_MC_PKT	RMON TX multicast packets
0x210	RMON_T_CRC_ALIGN	RMON TX packets with CRC/align error
0x214	RMON_T_UNDERSIZE	RMON TX packets < 64 bytes; good CRC
0x218	RMON_T_OVERSIZE	RMON TX packets > MAX_FL bytes; good CRC
0x21C	RMON_T_FRAG	RMON TX packets < 64 bytes; bad CRC
0x220	RMON_T_JAB	RMON TX packets > MAX_FL bytes; bad CRC
0x224	RMON_T_COL	RMON TX collision count
0x228	RMON_T_P64	RMON TX 64-byte packets
0x22C	RMON_T_P65TO127	RMON TX 65- to 127-byte packets
0x230	RMON_T_P128TO255	RMON TX 128- to 255-byte packets
0x234	RMON_T_P256TO511	RMON TX 256- to 511-byte packets
0x238	RMON_T_P512TO1023	RMON TX 512- to 1023-byte packets
0x23C	RMON_T_P1024TO2047	RMON TX 1024- to 2047-byte packets
0x240	RMON_T_P_GTE2048	RMON TX packets with > 2048 bytes
0x244	RMON_T_OCTETS	RMON TX octets
0x248	IEEE_T_DROP	Frames counted incorrectly

Table 16-9. MIB Counters (Offset 200–3FF) (Sheet 2 of 3)

Address	Mnemonic	Description
0x24C	IEEE_T_FRAME_OK	Frames transmitted OK
0x250	IEEE_T_1COL	Frames transmitted with single collision
0x254	IEEE_T_MCOL	Frames transmitted with multiple collisions
0x258	IEEE_T_DEF	Frames transmitted after deferral delay
0x25c	IEEE_T_LCOL	Frames transmitted with late collision
0x260	IEEE_T_EXCOL	Frames transmitted with excessive collisions
0x264	IEEE_T_MACERR	Frames transmitted with TX FIFO underrun
0x268	IEEE_T_CSERR	Frames transmitted with carrier sense error
0x26C	IEEE_T_SQE	Frames transmitted with SQE error
0x270	T_FDXFC	Flow control pause frames transmitted
0x274	IEEE_T_OCTETS_OK	Octet count for frames transmitted without error
0x278–0x27C	Reserved	Reserved
0x280	RMON_R_DROP	Frames counted incorrectly
0x284	RMON_R_PACKETS	RMON RX packet count
0x288	RMON_R_BC_PKT	RMON RX broadcast packets
0x28C	RMON_R_MC_PKT	RMON RX multicast packets
0x290	RMON_R_CRC_ALIGN	RMON RX packets with CRC/align error
0x294	RMON_R_UNDERSIZE	RMON RX packets < 64 bytes; good CRC
0x298	RMON_R_OVERSIZE	RMON RX packets > MAX_FL bytes; good CRC
0x29C	RMON_R_FRAG	RMON RX packets < 64 bytes; bad CRC
0x2A0	RMON_R_JAB	RMON RX packets > MAX_FL bytes; bad CRC
0x2A4	RMON_R_RESVD_0	
0x2A8	RMON_R_P64	RMON RX 64-byte packets
0x2AC	RMON_R_P65TO127	RMON RX 65- to 127-byte packets
0x2B0	RMON_R_P128TO255	RMON RX 128- to 255-byte packets
0x2B4	RMON_R_P256TO511	RMON RX 256- to 511-byte packets
0x2B8	RMON_R_P512TO1023	RMON RX 512- to 1023-byte packets
0x2BC	RMON_R_P1024TO2047	RMON RX 1024- to 2047-byte packets
0x2C0	RMON_R_P_GTE2048	RMON RX packets with > 2048 bytes
0x2C4	RMON_R_OCTETS	RMON RX octets
0x2C8	IEEE_R_DROP	Frames counted incorrectly

Table 16-9. MIB Counters (Offset 200–3FF) (Sheet 3 of 3)

Address	Mnemonic	Description
0x2CC	IEEE_R_FRAME_OK	Frames received OK
0x2D0	IEEE_R_CRC	Frames received with CRC error
0x2D4	IEEE_R_ALIGN	Frames received with alignment error
0x2D8	IEEE_R_MACERR	Receive FIFO overflow count
0x2DC	R_FDXFC	Flow control pause frames received
0x2E0	IEEE_R_OCTETS_OK	Octet count for frames received without error
0x2E4–0x2FC	Reserved	Reserved
0x300–0x3FF	Reserved	Reserved

16.3.5 Register Descriptions

16.3.5.1 FEC ID Register (FEC_ID)

The FEC_ID register (FEC_ID) is a read-only register. The FEC_ID register is used to identify the FEC block and revision.

Register address: 000

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FEC_ID															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	DMA	FIFO	SMII	FEC_REV							
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 16-3. FEC_ID Register

Table 16-10. FEC_ID Field Descriptions

Field	Description
FEC_ID	Unique identifier for FEC 0000
DMA	DMA function included in the FEC 1 FEC includes DMA (DMA_CONTROL register contains DMA revision) 0 EC does not include DMA
FIFO	FIFO function included in the FEC 1 FEC includes a FIFO (FIFO_ID register contains the FIFO revision) 0 FEC does not include a FIFO
SMII	The Ethernet PHY interface configuration. 1 SMII (serial MII) interface. This 6-pin serial MII option requires that the MII_MODE bit of the R_CNTRL register is set equal to 1. 0 MII (18 pins) or 7-wire (select MII or 7-wire via the MII_MODE bit of the R_CNTRL register)
FEC_REV	Value identifies the revision of the FEC. 00 Initial revision

16.3.5.2 Interrupt Event Register (IEVENT)

When an event occurs that sets a bit in the interrupt event register, an interrupt is generated if the corresponding bit in the interrupt enable register (IMASK) is also set. The bit in the interrupt event register is cleared if a 1 is written to that bit position. Writing 0 has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts that may occur in normal operation are GRA, TFINT, TXB, RFINT, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LATE_COL and COL_RETRY_LIM. Interrupts resulting from internal errors are EBERR and XFIFO_UN.

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts because these errors are visible to network management via the MIB counters.

- HBERR — ieee_t_sqe
- BABR — rmon_r_oversize (good CRC), rmon_r_jab (bad CRC)
- BABT — rmon_t_oversize (good CRC), rmon_t_jab (bad CRC)
- LATE_COL — ieee_t_lcol
- COL_RETRY_LIM — ieee_t_excol
- XFIFO_UN — ieee_t_macerr

Address 004

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HBERR	BABR	BABT	GRA	TFINT	TXB	RFIN T	RXB	MII	EBERR	LATE_COL	COL_RETR Y_LI M	XFIFO_U N	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 16-4. IEVENT Register

Table 16-11. IEVENT Field Descriptions

Field	Description
HBERR	Heartbeat error. This interrupt indicates HBC is set in the X_CNTRL register and the COL input was not asserted within the heartbeat window following a transmission.
BABR	Babbling receive error. This bit indicates a frame was received with length in excess of R_CNTRL.MAX_FL bytes.
BABT	Babbling transmit error. This bit indicates the transmitted frame length has exceeded R_CNTRL.MAX_FL bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffer(s). Truncation does not occur.
GRA	Graceful stop complete. This interrupt is asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. 1 A graceful stop, which was initiated by the setting of the GTS bit of the X_CNTRL register, is now complete. 2 A graceful stop, which was initiated by the setting of the FC_PAUSE bit of the X_CNTRL register, is now complete. 3 A graceful stop, which was initiated by the reception of a valid full-duplex flow control pause frame, is now complete. Refer to Section 16.6.4, "Full-Duplex Flow Control" .
TFINT	Transmit frame interrupt. This bit indicates a frame has been transmitted and the last corresponding buffer descriptor has been updated.
TXB	Transmit buffer interrupt. This bit indicates a transmit buffer descriptor that had the R bit set in its status word has been updated.

Table 16-11. IEVENT Field Descriptions (continued)

Field	Description
RFINT	Receive frame interrupt. This bit indicates a frame has been received and the last corresponding buffer descriptor has been updated.
RXB	Receive buffer interrupt. This bit indicates a receive buffer descriptor that had the E bit set in its status word has been updated.
MII	MII interrupt. This bit indicates the MII has completed the requested data transfer.
EBERR	Ethernet bus error. This bit indicates a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, the ETHER_EN of the of the ECNTRL register is cleared, halting frame processing by the FEC.
LATE_COL	Late collision. This bit indicates a collision occurred beyond the collision window (slot time) in half-duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
COL_RETRY_LIM	Collision retry limit. This bit indicates a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame commences. This situation can only occur in half-duplex mode.
XFIFO_UN	Transmit FIFO underrun. This bit indicates the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.

16.3.5.3 Interrupt Mask Register (IMASK)

The interrupt mask register provides control over which possible interrupt events are allowed to generate an actual interrupt. All implemented bits in this CSR are read/write. This register is cleared upon a hardware reset. If the corresponding bits in both the IEVENT and IMASK registers are set, the interrupt is signalled to the CPU. The interrupt signal remains asserted until a 1 is written to the IEVENT bit (write 1 to clear) or a 0 is written to the IMASK bit.

Fast Ethernet Controller (FEC)

Register address: 008

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HBEEN	BREN	BTEN	GRAEN	TFIEN	TBIEN	RFIEN	RBIEN	MIIEN	EBERREN	LCEN	CRLEN	XFUNEN	Reserved		0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-5. IMASK Register

Table 16-12. IMASK Field Descriptions

Field	Description
HBEEN	Heartbeat error interrupt enable
BREN	Babbling receiver interrupt enable
BTEN	Babbling transmitter interrupt enable
GRAEN	Graceful stop interrupt enable
TFIEN	Transmit frame interrupt enable
TBIEN	Transmit buffer interrupt enable
RFIEN	Receive frame interrupt enable
RBIEN	Receive buffer interrupt enable
MIIEN	MII interrupt enable
EBERREN	Ethernet controller bus error enable
LCEN	Late collision enable
CRLEN	Collision retry limit enable
XFUNEN	Transmit FIFO underrun enable

16.3.5.4 CSR Receive Descriptor Active Register (R_DES_ACTIVE)

The CSR descriptor active register is a command register that should be written to indicate the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the E bit set).

The R_DES_ACTIVE bit is set when the register is written. This is independent of the data actually written. When set, the FEC polls the receive descriptor ring and processes receive frames, provided ECNTRL.ETHER_EN is also set. After the FEC polls a receive descriptor whose ownership bit is not set, the FEC clears the R_DES_ACTIVE bit and ceases receive descriptor ring polling until you set the bit again, signifying additional descriptors have been placed into the receive descriptor ring.

The R_DES_ACTIVE register is cleared at reset and by the clearing of the ETHER_EN bit of the ECNTRL register.

Register address: 010

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	R_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-6. R_DES_ACTIVE Register

Table 16-13. R_DES_ACTIVE Field Descriptions

Field	Description
R_DES_ACTIVE	This bit is set to 1 when this register is written, regardless of the value written. It is cleared by the FEC device when no additional ready descriptors remain in the receive ring.

16.3.5.5 CSR Transmit Descriptor Active Register (X_DES_ACTIVE)

The CSR descriptor active register is a command register that should be written to indicate the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the R bit set in the buffer descriptor).

The X_DES_ACTIVE bit is set when the register is written. This is independent of the data actually written. When set, the FEC polls the transmit descriptor ring and process transmit frames, provided ETHER_EN bit of the ECNTL register is also set. After the FEC polls a transmit descriptor whose ownership bit is not set, the FEC clears the X_DES_ACTIVE bit and ceases transmit descriptor ring polling until you set the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The X_DES_ACTIVE register is cleared at reset and by the clearing of the ETHER_EN bit of the ECNTL register.

Register address: 014

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	X_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-7. X_DES_ACTIVE Register

Table 16-14. X_DES_ACTIVE Field Descriptions

Field	Description
X_DES_ACTIVE	This bit is set to 1 when this register is written, regardless of the value written. It is cleared by the FEC device when no additional ready descriptors remain in the transmit ring.

16.3.5.6 Ethernet Control Register (ECNTRL)

The Ethernet control register (ENCNTRL) is a read/write user register; however, some fields may be altered by hardware as well. The ENCNTRL register enables/disables the FEC. The Reserved bits must be cleared.

Register address: 024

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	Reserved	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	Reserved	ETHER_EN	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-8. ECNTRL Register

Table 16-15. ECNTRL Field Descriptions

Field	Description
ETHER_EN	Ethernet enable. When this bit is set, the fast Ethernet controller is enabled, and reception and transmission is possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any frame currently being transmitted. The buffer descriptor(s) for an aborted transmit frame are not updated following deassertion of ETHER_EN. When ETHER_EN is deasserted, the DMA, buffer descriptor, and FIFO control logic are reset, including buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions: <ul style="list-style-type: none"> If ecntrl.RESET is written to a 1 by software, ETHER_EN is cleared If error conditions occur, causing the EBERR bit of the IEVENT register to set, ETHER_EN is cleared.
RESET	Ethernet controller reset. When this bit is set, the equivalent of a hardware reset is performed, but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately eight clock cycles after reset is written with a 1. It's recommended to read ECNTRL register back and this can give enough time to go through reset sequence

16.3.5.7 MII Management Frame Register (MII_DATA)

The MII_DATA register does not reset to a defined value. The MII_DATA register communicates with the attached MII-compatible PHY device, providing read/write access to MII registers. Performing a write to the MII_DATA register causes a management frame to be sourced, unless the MII_SPEED register has been programmed to 0. Writing to MII_DATA when MII_SPEED equals 0, and the MII_SPEED register is then written to a non-zero value, an MII frame is generated with the data previously written to the MII_DATA register. This allows MII_DATA and MII_SPEED to be programmed in either order if MII_SPEED is currently zero.

Register address: 040

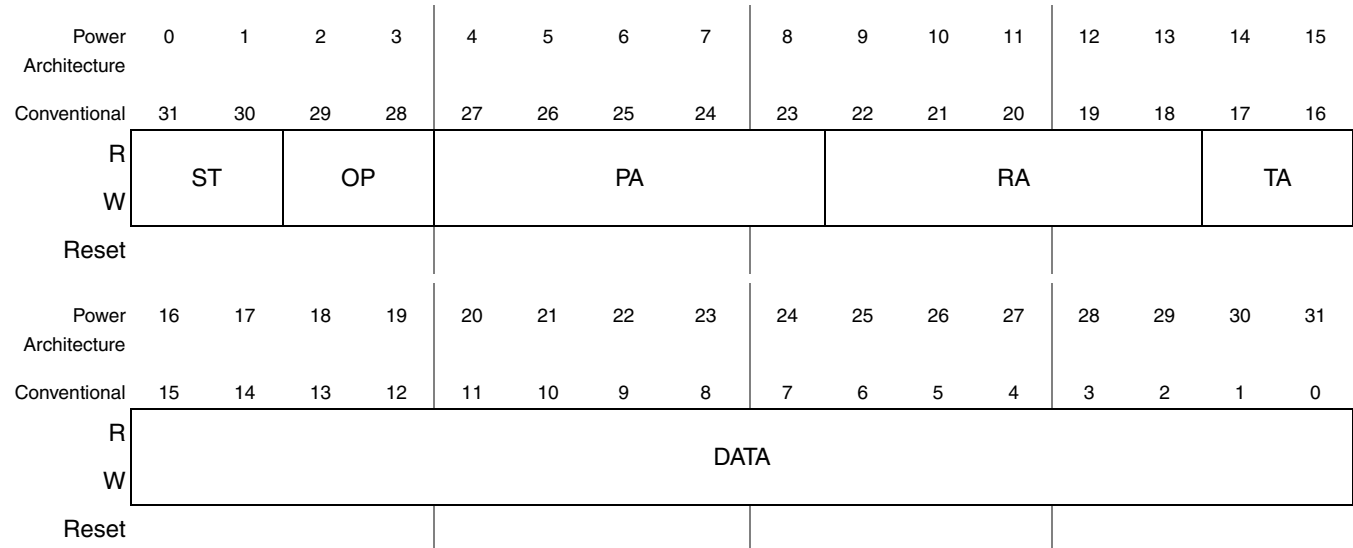


Figure 16-9. MII_DATA Register

Table 16-16. MII_DATA Field Descriptions

Field	Description
ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
OP	Operation code. This field must be programmed to 10 (read) or 01 (write) to generate a valid MII management frame. A value of 11 produces read frame operation, while a value of 00 produces write frame operation, but these frames are not MII-compliant.
PA	PHY address. This field specifies one of up to 32 attached PHY devices.
RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.
TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

Write the MII_DATA register to perform a read or write operation on the MII management interface. To generate a valid read or write management frame, the ST field must be written with a 01 pattern, the OP field must be written with a 01 (management register write frame) or 10 (management register read frame), and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated,

but does not comply with the IEEE 802.3 MII definition. When op field equals 1x, it produces a read-frame operation, while op equals 0x produces a write-frame operation.

To generate an 802.3-compliant MII management interface write frame (write to a PHY register), write {01 01 PHYAD REGAD 10 DATA} to the MII_DATA register. Writing this pattern causes the control logic to shift out the data in the MII_DATA register following a preamble generated by the control state machine. During this time, the contents of the MII_DATA register is altered as the contents are serially shifted, and are unpredictable if read by the user. The MII interrupt is generated after the write management frame operation has been completed. At this time, the contents of the MII_DATA register match the original value written.

To generate an MII management interface read frame (read a PHY register), write {01 10 PHYAD REGAD 10 XXXX} to the MII_DATA register (the content of the data field is a don't care). Writing this pattern causes the control logic to shift out the data in the MII_DATA register following a preamble generated by the control state machine. During this time, the contents of the MII_DATA register are altered as the contents are serially shifted and are unpredictable if read. The MII interrupt is generated after the read management frame operation has completed. At this time, the contents of the MII_DATA register match the original value written, except for the data field, where contents have been replaced by the value read from the PHY register.

If the MII_DATA register is written while frame generation is in progress, the frame contents are altered. Software should use the MII_STATUS register and/or the MII_DATAIO_COMPL interrupt to avoid writing to the MII_DATA register while frame generation is in progress.

16.3.5.8 MII Speed Control Register (MII_SPEED)

The MII_SPEED register provides control of the MII clock (MDC pin) frequency, allows dropping the preamble on the MII management frame, and provides observability (intended for manufacturing test) of an internal counter used in generating the MDC clock signal.

Register address: 044

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	DIS_PRE-AMBLE	MII_SPEED						0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 16-10. MII_SPEED Register

Table 16-17. MII_SPEED Field Descriptions

Field	Description
DIS_PREAMBLE	Asserting this bit causes preamble (32 ones) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (MDC) relative to system clock. A value of 0 in this field turns off the MDC and leaves it in a low-voltage state. Any non-zero value results in the MDC frequency of $1/(mii_speed * 2)$ of the system clock frequency.

To be compliant with the IEEE MII specification, the MII_SPEED field must be programmed with a value that provides an MDC frequency of less than or equal to 2.5 MHz. The MII_SPEED bit must be set to a non-zero value to source a read or write management frame. After the management frame is complete, the MII_SPEED register may optionally be set to 0 to turn off the MDC. The MDC generated has a 50% duty cycle, except when MII_SPEED is changed during operation (the change takes effect following a rising or falling edge of MDC).

If the system clock is 25 MHz, programming this register to 0x0000_0005 results in an MDC frequency of $25 \text{ MHz} * 1/(2*5) = 2.5 \text{ MHz}$. [Table 16-18](#) shows optimum values for MII_SPEED as a function of system clock frequency.

Table 16-18. Programming Examples for MII_SPEED Register

IPS Clock Frequency	II_SPEED (Field in Register)	MDC Frequency
25 MHz	0x5	2.5 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.5 MHz
50 MHz	0xA	2.5 MHz
66MHz	0xE	2.36MHz
83MHz	0x11	2.44MHz

16.3.5.9 MIB Control Register (MIB_CONTROL)

The MIB_CONTROL register is a read/write register that provides control of and observes the state of the MIB block. If it is necessary to disable the MIB block operation, this register is accessed by user software. For example, to clear all MIB counters in RAM, disable the MIB block, clear all the MIB RAM locations, and then enable the MIB block. The MIB_DISABLE bit is reset to 1.

Register address: 064

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MIB_DISABLE	MIB_IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-11. MIB_CONTROL Register

Table 16-19. MIB_CONTROL Field Descriptions

Field	Description
MIB_DISABLE	This is a read/write control bit. If set, the MIB logic halts and does not update any MIB counters.
MIB_IDLE	This is a read-only status bit. If set, the MIB block is not currently updating any MIB counters.

16.3.5.10 Receive Control Register (R_CNTRL)

The R_CNTRL register controls the operational mode of the receive block and should only be written when the ETHER_EN bit of the ECNTRL register equals 0 (initialization time).

Register address: 084

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PRO M	MII_ MOD E	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1


 = Unimplemented or Reserved

Figure 16-12. R_CNTRL Register

Table 16-20. R_CNTRL Field Descriptions

Field	Description
MAX_FL	Maximum frame length. This is a user read/write field that resets to decimal 1518. Length is measured starting at DA (destination address) and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes a BABT interrupt to occur. Receive frames longer than MAX_FL cause a BABR interrupt to occur and set the LG bit in the end-of-frame buffer descriptor. The recommended default value to be programmed is 1518 or 1522 (if VLAN tags are supported).
FCE	Flow control enable. If asserted, the receiver detects pause frames. The transmitter stops transmitting data frames for a given duration when pause frames are detected.
BC_REJ	Broadcast frame reject. If asserted, frames with DA = FFFF_FFFF_FFFF are rejected unless the PROM bit is set. If BC_REJ and PROM equal 1 individually, frames with broadcast DA are accepted and the MISS (M) bit is set in the receive buffer descriptor.
PROM	Promiscuous mode. All frames are accepted, regardless of address matching.
MII_MODE	Selects external interface mode. Setting this bit to 1 selects MII mode; setting this bit equal to 0 selects 7-wire mode (used only for serial 10 Mbps). This bit controls the interface mode for both transmit and receive blocks.

Register address: 084

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PRO M	MII_MOD E	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 16-12. R_CNTRL Register

Table 16-20. R_CNTRL Field Descriptions (continued)

Field	Description
DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full-duplex or to monitor transmit activity in half-duplex mode). 1 Disable reception of frames while transmitting (normally used for half-duplex mode).
LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the TX_CLK when loop is asserted. DRT must be set to 0 when asserting loop.

16.3.5.11 R_HASH Register

This read-only register provides address recognition information from the receive block about the frame currently being received. This field is read by the FEC. These bits provide the FEC with information used in the address recognition subroutine.

Register address: 088

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FCE_DC	MULT-CAST	HASH						0	0	0	0	0	0	0	0
W																
Reset																
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset																

= Unimplemented or Reserved

Figure 16-13. R_HASH Register

Table 16-21. R_HASH Field Descriptions

Field	Description
FCE_DC	This is a read-only view of the FCE bit in the R_CNTRL register.
MULTICAST	This bit is set if the current receive frame contained a multicast destination address (the least significant bit of the DA was set). It is cleared if the current receive frame does not correspond to a multicast address.
HASH	Corresponds to the hash value of the current receive frame's destination address. The hash value is a 6-bit field extracted from the least significant portion of the CRC register.

NOTE

FCE_DC, MULT CAST, and HASH are not affected by PORESET or HRESET.

16.3.5.12 Transmit Control Register (X_CNTRL)

This register is read/write and is written to configure the transmit block. This register is cleared at system reset. Bits FDEN and HBC should be modified only when the ETHER_EN bit of the ECNTRL register equals 0.

Register address: 0C4

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	RFC_PAUSE	TFC_PAUSE	FDE	HBC	GST
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-14. X_CNTRL Register

Table 16-22. X_CNTRL Field Descriptions

Field	Description
RFC_PAUSE	This read-only status bit is asserted when a full-duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
TFC_PAUSE	This bit is asserted to transmit a pause frame. When this bit is set, the MAC stops transmission of data frames after the current transmission is complete. At this time, the GRA interrupt in the INTR_EVENT register is asserted. With transmission of data frames stopped, the MAC transmits a MAC control pause frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. If the transmitter is paused due to assertion of GTS or reception of a pause frame, the MAC may continue to transmit a MAC control pause frame.
FDEN	Full-duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should be modified only when ETHER_EN bit of the ECNTRL register is deasserted.

Register address: 0C4

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	RFC_PAUSE	TFC_PAUSE	FDE_N	HBC	GST
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-14. X_CNTRL Register

Table 16-22. X_CNTRL Field Descriptions (continued)

Field	Description
HBC	Heartbeat control. If set, the heartbeat check is performed following the end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should be modified only when the ETHER_EN bit of the ECNTRL register is deasserted.
GTS	Graceful transmit stop. When this bit is set, the MAC stops transmission after completion of any frame currently being transmitted, and the GRA interrupt in the INTR_EVENT register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission is complete, a restart can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS equals 1, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that are transmitted when GTS is reasserted. To avoid this, deassert ETHER_EN bit of the ECNTRL register following the GRA interrupt.

16.3.5.13 Physical Address Low (PADDR1)

This register contains the lower 32 bits (bytes 0, 1, 2, 3) of the 48-bit address used in the address recognition process to compare with the destination address (DA) field of receive frames with an individual DA. In addition, this register is used in bytes 0–3 of the 6-byte source address field when transmitting pause frames. This register is not reset and must be initialized by the user.

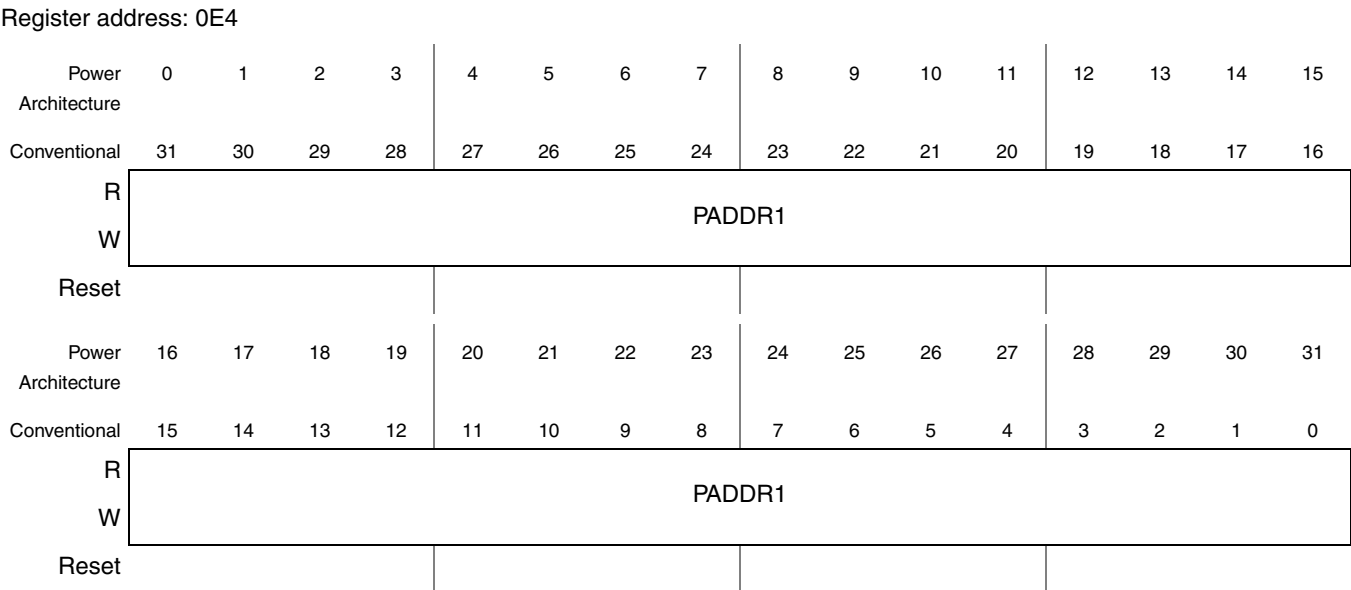


Figure 16-15. PADDR1 Register

Table 16-23. PADDR1 Field Descriptions

Field	Description
PADDR1	This field comprises bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8), and 3 (bits 7:0) of the 6-byte individual address to be used for exact match, and the source address field in pause frames.

16.3.5.14 Physical Address High (PADDR2)

This register contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte source address field when transmitting pause frames. Bits 16:31 of PADDR2 contain a constant-type field (hex 8808) used for transmission of pause frames. This register is not reset and bits 0:15 must be initialized.

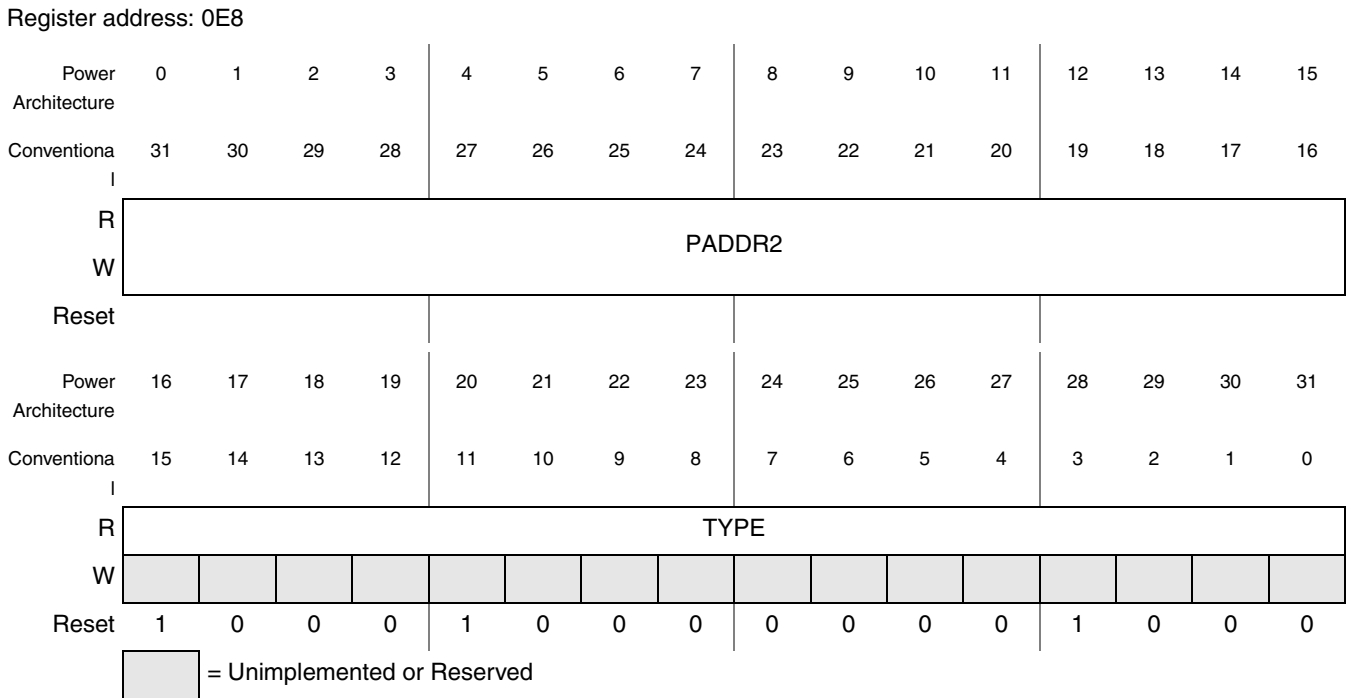


Figure 16-16. PADDR2 Register

Table 16-24. PADDR2 Field Descriptions

Field	Description
PADDR2	This field comprises bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address to be used for an exact match, and the source address field in pause frames.
TYPE	This is the type field in pause frames. These 16-bits are a constant value of hex 8808.

16.3.5.15 Opcode/Pause Duration Register (OP_PAUSE)

The OP_PAUSE register is read/write accessible. This register contains the 16-bit opcode, and 16-bit pause duration fields used in transmission of a pause frame. The opcode field is a constant value, hex 0001. When another node detects a pause frame, that node pauses transmission for the duration specified in the pause duration field. This register is not reset and must be initialized.

Register address: 0EC

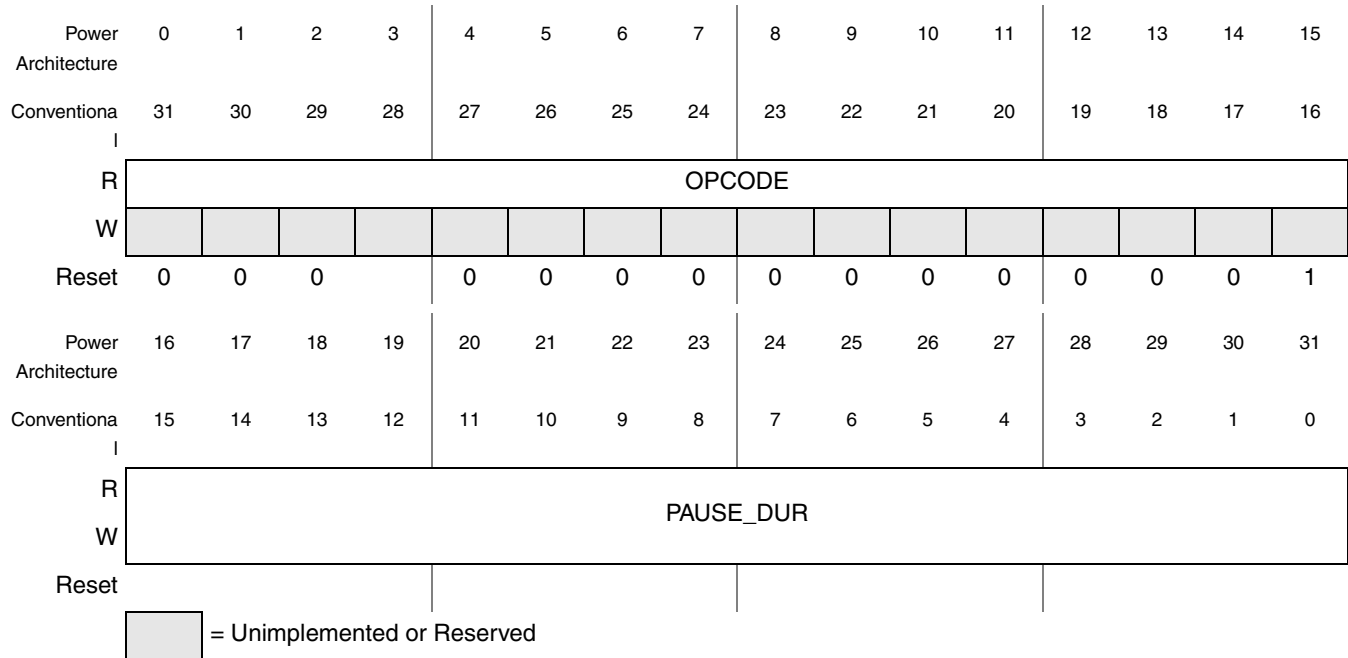


Figure 16-17. OP_PAUSE Register

Table 16-25. OP_PAUSE Field Descriptions

Field	Description
OPCODE	This is the opcode field used in pause frames. These bits are a constant, hex 0001.
PAUSE_DUR	This is the pause duration field used in pause frames.

16.3.5.16 Descriptor Individual Address 1 (IADDR1)

This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for a possible match between the DA field of receive frames and an individual DA. This register is not reset and must be initialized.

Register address: 118

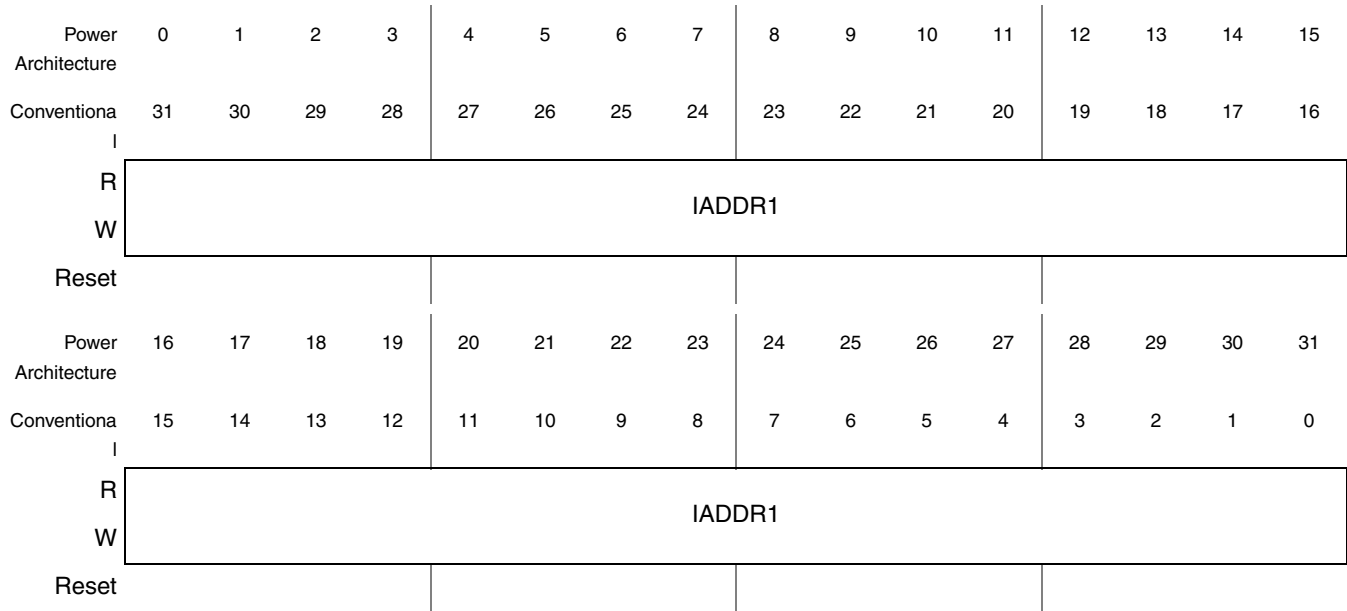


Figure 16-18. IADDR1 Register

Table 16-26. IADDR1 Field Descriptions

Field	Description
IADDR1	This field is the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

16.3.5.17 Descriptor Individual Address 2 (IADDR2)

This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match between the DA field of receive frames and an individual DA. This register is not reset and must be initialized.

Register address: 11C



Figure 16-19. IADDR2 Register

Table 16-27. IADDR2 Field Descriptions

Field	Description
IADDR2	This field is the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 63. Bit 0 of IADDR2 contains hash index bit 32.

16.3.5.18 Descriptor Group Address 1 (GADDR1)

This register contains the upper 32 bits of the 64-bit hash address table used in the address recognition process for receive frames with a multicast address. This register is not affected by PORESET or HRESET and must be initialized.

Register address: 120

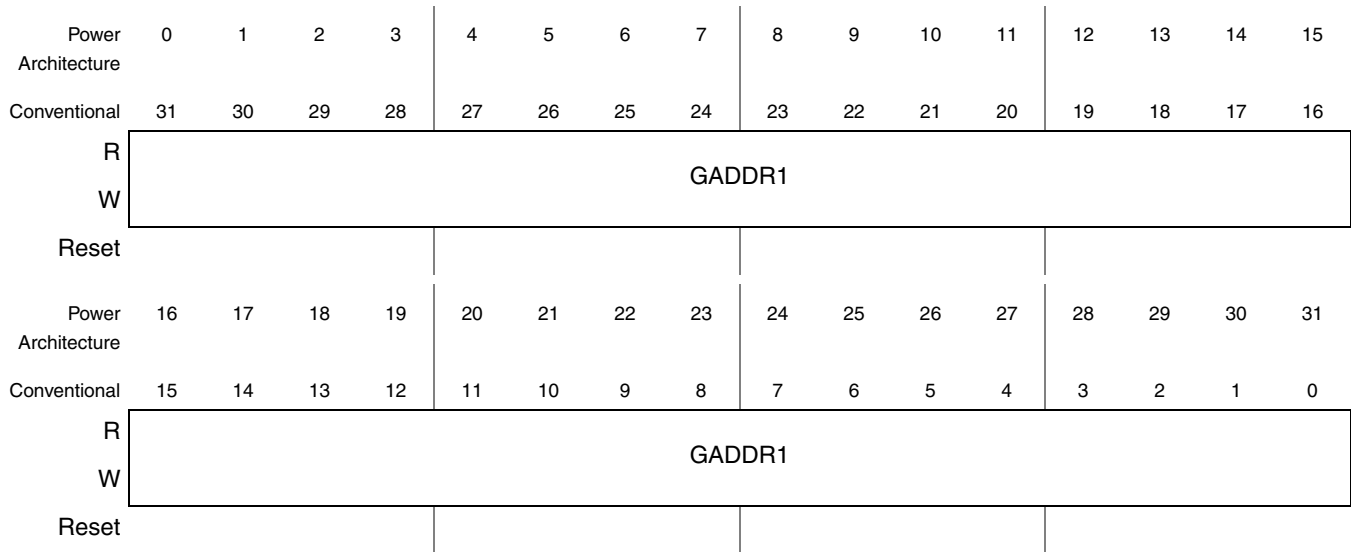


Figure 16-20. GADDR1 Register

Table 16-28. GADDR1 Field Descriptions

Field	Description
GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table address used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

16.3.5.19 Descriptor Group Address 2 (GADDR2)

The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register is not affected by PORESET or HRESET and must be initialized.

Register address: 124



Figure 16-21. GADDR2 Register

Table 16-29. GADDR2 Field Descriptions

Field	Description
GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

16.3.5.20 FIFO Transmit FIFO Watermark (X_WMRK)

The X_WMRK register is a 2-bit read/write register programmed to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows you to minimize transmit latency (X_WMRK = 0X) or allow for larger bus access latency (X_WMRK = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the X_WMRK field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel). The X_WMRK register resets to zero.

The logic and definition of this register may need to change for a specific instantiation/application of the FEC to be compatible with specific FIFO/system bus access latency requirements.

Register address: 144

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X_WMRK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-22. X_WMRK Register

Table 16-30. X_WMRK Field Descriptions

Field	Description
X_WMRK	Transmit FIFO watermark. Frame transmission begins when the number of bytes selected by this field have been written into the transmit FIFO if an end-of-frame has been written to the FIFO or if the FIFO is full before the selected number of bytes have been written. The options are: 0X 64 bytes written to xFIFO 10 128 bytes written to xFIFO 11 192 bytes written to xFIFO

16.3.5.21 FIFO Receive Bound Register (R_BOUND)

The R_BOUND register is an 8-bit register that can be read to determine the upper address bound of the FIFO RAM. The highest address of FIFO_RAM is R_BOUND-1. Drivers can use the value of R_BOUND, along with the R_FSTART register value, to appropriately divide the available FIFO RAM between the transmit and receive data paths.

The R_BOUND register is read-only.

Register address: 14C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	1	R_BOUND								0	0
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 16-23. R_BOUND Register

Table 16-31. R_BOUND Field Descriptions

Field	Description
R_BOUND	Read-only; determine the highest valid FIFO RAM address.

16.3.5.22 FIFO Receive Start Register (R_FSTART)

The R_FSTART register is an 8-bit register programmed to indicate the starting address of the receive R_FSTART marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from 0 to R_FSTART-4. The receive FIFO uses addresses from R_FSTART to R_BOUND-1, inclusive.

The R_FSTART register is initialized by hardware at reset. Write R_FSTART to change the default value.

Register address: 150

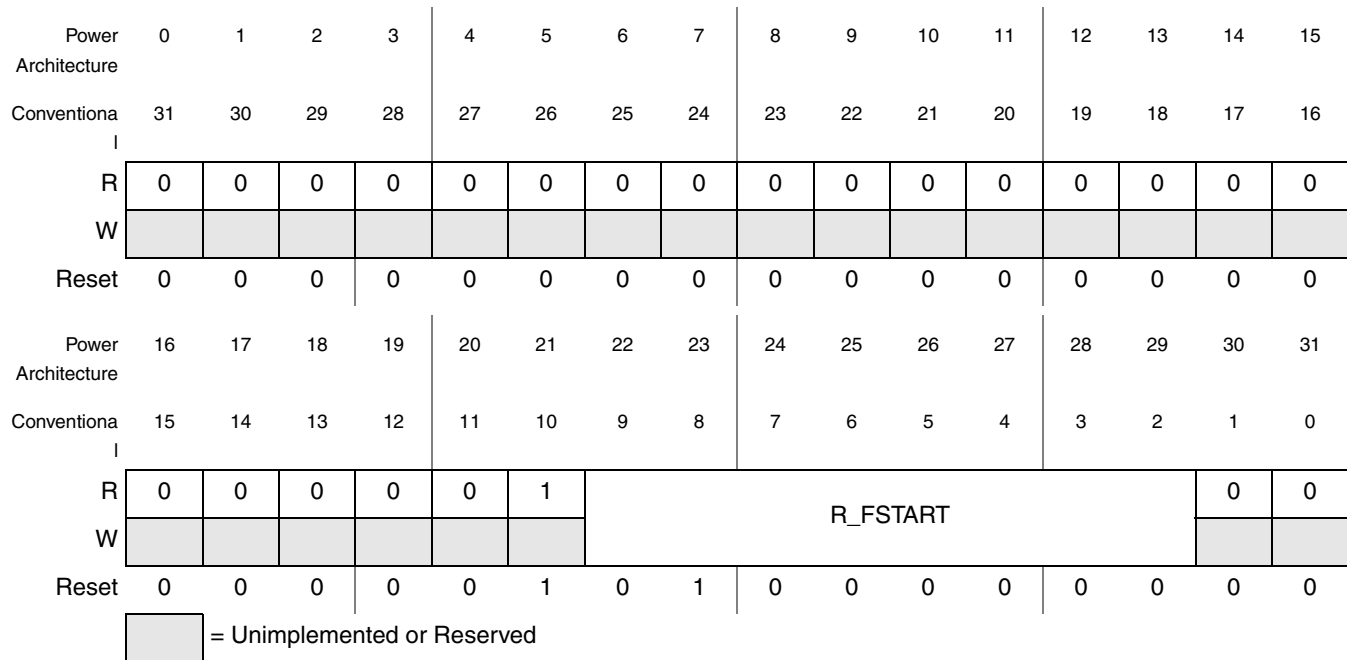


Figure 16-24. R_FSTART Register

Table 16-32. R_FSTART Field Descriptions

Field	Description
R_FSTART	This is the address of the first receive FIFO location. It acts as a delimiter between the receive and transmit FIFOs.

16.3.5.23 Beginning of Receive Descriptor Ring (R_DES_START)

This register is a pointer to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; Write bits 30, 31 to 0. It is strongly recommended to be quad word-aligned (evenly divisible by 16) to get better system performance; write bits 28, 29, 30, and 31 to 0.

This register is not reset and must be initialized prior to operation.

Register address: 180

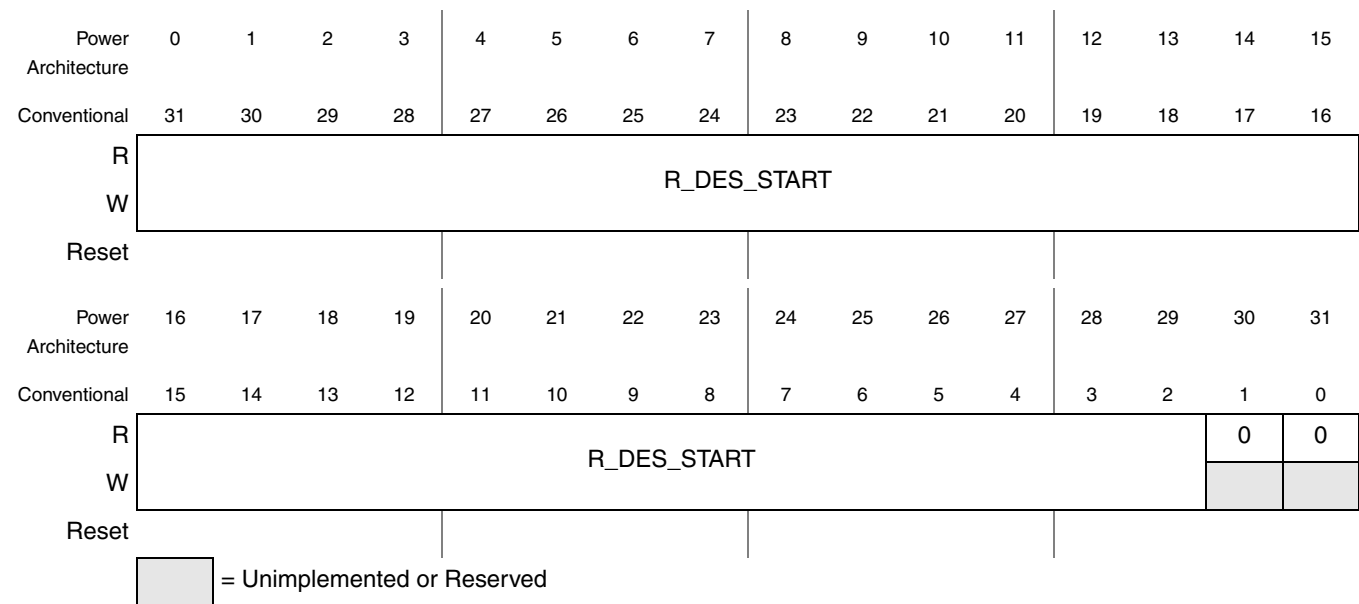


Figure 16-25. R_DES_START Register

Table 16-33. R_DES_START Field Descriptions

Field	Description
R_DES_START	This is a pointer to the start of the receive buffer descriptor queue.

16.3.5.24 Beginning of Transmit Descriptor Ring (X_DES_START)

This register is a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned, write bits 30 ,31 to 0. It is strongly recommended to be quad word-aligned (evenly divisible by 16) to get better system performance; write bits 28, 29, 30 and 31 to 0. This register is not reset and must be initialized prior to operation.

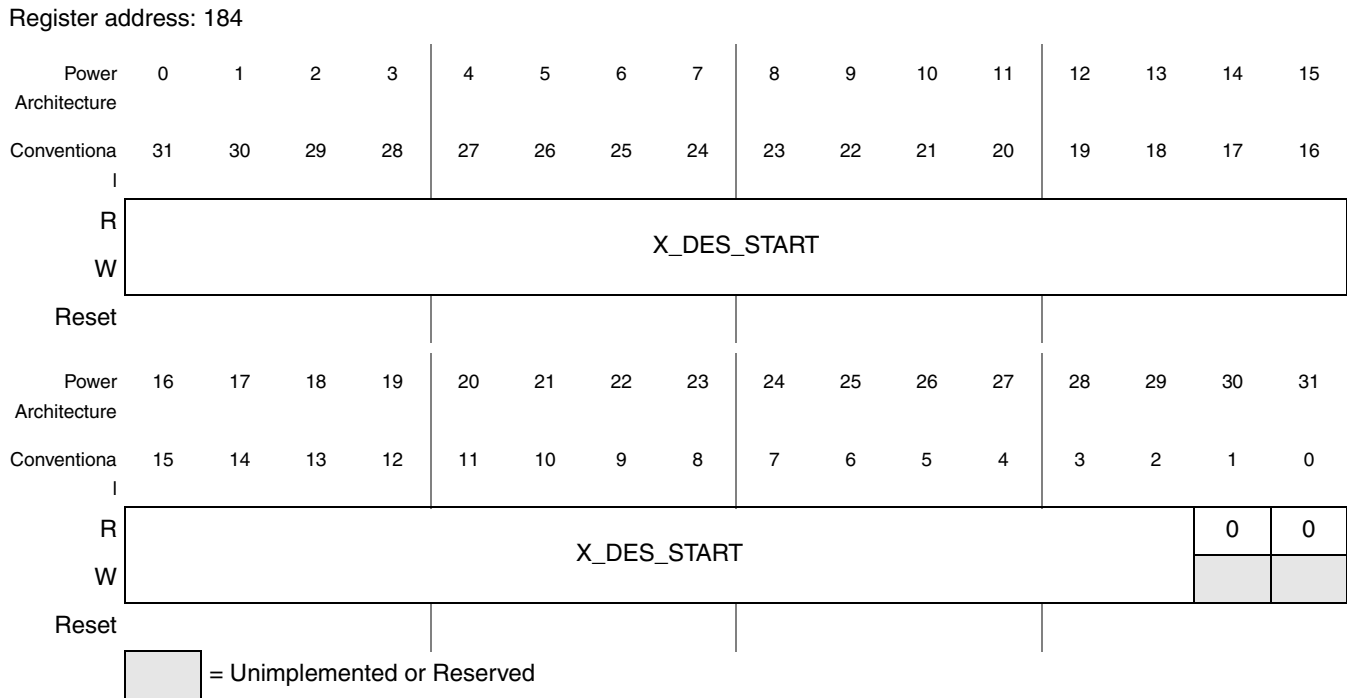


Figure 16-26. X_DES_START Register

Table 16-34. X_DES_START Field Descriptions

Field	Description
X_DES_START	This is a pointer to the start of the transmit buffer descriptor queue.

16.3.5.25 Receive Buffer Size Register (R_BUFF_SIZE)

The R_BUFF_SIZE registers dictates the maximum size of all receive buffers. Only bits 21 – 27 are used, because receive frames are truncated at 2k-1 bytes. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum-sized frame per buffer, R_BUFF_SIZE must be set to R_CNTRL.MAX_FL or larger. The R_BUFF_SIZE must be evenly divisible by 16. To ensure this, bits 3-0 are forced to 0. To minimize bus utilization (descriptor fetches), it is recommended that R_BUFF_SIZE be greater than or equal to 256 bytes.

The R_BUFF_SIZE register is not affected by reset and must be initialized.

Register address: 188

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset																
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	R_BUFF_SIZE							0	0	0	0
W																
Reset																


 = Unimplemented or Reserved

Figure 16-27. R_BUFF_SIZE Register Definition

Table 16-35. R_BUFF_SIZE Field Descriptions

Field	Description
R_BUFF_SIZE	This is the receive buffer size.

16.3.5.26 DMA Function Control Register (DMA_CONTROL)

The DMA_CONTROL register contains the function code and byte order fields used during each transfer between the DMA and the FEC interface. These bits can be written/read by the user. This register should be programmed only when the ETHER_EN bit of the ECNTRL register equals 0.

Register address: 1F4

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DATA_BO	DESC_BO	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset			0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	DMA_REV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 16-28. DMA_CONTROL Register

Table 16-36. DMA_CONTROL Field Descriptions

Field	Description
DATA_BO	The byte order control for data DMA transfers. 0 Little endian (bytes 0 and 3 swapped, bytes 1 and 2 swapped) 1 Big endian (no byte swapping) Note: This bit is not affected by reset and must be initialized before using the DMA controller.
DESC_BO	The byte order control for descriptor DMA transfers. 0 Little endian (bytes 0 and 3 swapped, bytes 1 and 2 swapped) 1 Big endian (no byte swapping) Note: The DATA_BO and DESC_BO fields are muxed to generate the IPM_BO signal on the master interface. Note: This bit is not affected by reset and must be initialized before using the DMA controller.
DMA_REV	DMA revision; read only

16.4 Initialization Information

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC (shown in [Figure 16-1](#)), and what locations must be initialized prior to enabling FEC.

16.4.1 Initialization (Prior to Asserting ETHER_EN)

Initialize portions of the FEC prior to setting the ETHER_EN bit of the ECNTRL register. The exact values depend on the particular application. The sequence is not important. Ethernet MAC registers requiring initialization are defined in [Table 16-37](#).

Table 16-37. Initialization Before ETHER_EN

Description
Start FEC Clock (SCCR.1.FEC_EN)
Initialize IMASK
Clear IEVENT (write FFFF_FFFF)
X_WMRK (optional)
IADDR2/IADDR1
GADDR1/GADDR2
PADDR1/PADDR2
OP_PAUSE (only needed for FDX flow control)
R_CNTRL
X_CNTRL
MII_SPEED (optional)
Clear MIB_RAM (locations 200 – 2E3)

FEC FIFO/DMA registers requiring initialization are defined in [Table 16-38](#).

Table 16-38. FEC Initialization (Before ETHER_EN)

Description
Initialize R_FSTART (optional)
Initialize R_BUFF_SIZE
Initialize R_DES_START
Initialize X_DES_START
Initialize DMA_CONTROL
Initialize (empty) transmit descriptor ring
Initialize (empty) receive descriptor ring

16.4.1.1 Descriptor Controller Initialization

In the fast Ethernet controller, the descriptor control RISC initializes some registers after ETHER_EN is asserted. After the descriptor controller initialization sequence is complete, the hardware is ready for operation.

The following table shows FEC RISC(shown in [Figure 0-1](#)) initialization operations.

Table 16-39. Descriptor Controller Initialization

Description
Initialize Backoff random number seed
Activate receiver
Activate transmit

The following table shows FEC RISC initialization operations specific to FEC.

Table 16-40. Descriptor Controller Initialization (FEC-Specific)

Description
Clear transmit FIFO X_LAG, X_READ, X_WRITE = 0
Clear receive FIFO R_LAG, R_READ, R_WRITE = R_FSTART
Initialize transmit ring pointer RDES_ADDR = R_DES_START
Initialize receive ring pointer XDES_ADDR = X_DES_START
Initialize FIFO count registers R_COUNT = X_COUNT = 0

16.4.1.2 Initialization (After Asserting ETHER_EN)

After asserting the ETHER_EN bit of the ECNTRL register, you can set up the buffer/frame descriptors and write to X_DES_ACTIVE and R_DES_ACTIVE.

16.5 Buffer Descriptors

16.5.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. A buffer descriptor (BD) that contains a starting address (pointer) and data length for the buffer is associated with each buffer. In addition to pointing to the buffer, the most significant bit of the BD is an ownership bit, which defines the current state of the buffer. Other bits in the buffer descriptor are used to communicate status/control information between the Ethernet MAC and the driver. To permit maximum user flexibility, the BDs used by the FEC DMA engines are also located in external memory.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the R/E (ownership) bit in the most significant word of the transmit (receive) buffer descriptor produces the buffer. A software write to either the X_DES_ACTIVE or R_DES_ACTIVE register tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the R(E) bit is cleared by hardware to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ETHER_EN signal operates as a reset to the BD/DMA logic. When ETHER_EN is deasserted, the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software (write 0x0000_0000 to the most significant word of buffer descriptor) before the ETHER_EN bit is set.

The buffer descriptors operate as a ring. R_DES_START defines the starting address for receive BDs and X_DES_START defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the wrap (W) bit. When set, w indicates that the next descriptor in the ring is at the location pointed to by R_DES_START and X_DES_START for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; it is strongly recommended they are made 128-bit aligned.

16.5.1.1 Driver/DMA Operation with Transmit BDs

Typically, a transmit frame is divided between multiple buffers. An example is to have an Ethernet/802.3 header in the first buffer, an IP header in a second buffer, a TCP header in a third buffer, and an application payload in the last buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type fields), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD, which must be set by the driver. When the DMA of the transmit frame is complete, the DMA controller appends a control word to the frame. The requirement for the control word is that the TC and ABC bits must be in the same position, as defined by the transmit BD. The simplest solution is to copy the most significant 32 bits of the transmit BD into the transmit FIFO at the end of the frame.

In a typical end station application, the TC bit always equals 1. For a switch/router application, the TC bit may be 1 or 0, depending on what type of port the frame arrived on and whether the frame contents were modified. The append bad CRC (ABC) bit is 0 unless an error has occurred (for example, a data parity error during DMA transfer) that results in data corruption.

The driver (TxBD software producer) should set up TxBDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length, and control (W, L, TC, ABC), and then the ownership (R) bits should be set equal to 1 in reverse order (BD 3, BD 2, BD 1) to ensure the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the second was made available, potentially causing a transmit FIFO underrun.

In the FEC, the driver notifies the DMA that new transmit frame(s) are available by writing to the X_DES_ACTIVE register. When data is written to this register (data value is not significant), the FEC RISC tells the DMA to read the next transmit BD in the ring. After the start, the RISC + DMA continue reading and interpreting transmit BDs in order and DMA the associated buffers until a transmit BD is encountered with the R bit cleared to 0. At this point, the FEC polls this BD one more time. If the R bit equals 0 a second time, the RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to X_DES_ACTIVE.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R (ownership) bit, indicating that the hardware consumer is finished with the buffer. A second driver task (TxBD software consumer) processes the transmit descriptor ring and return buffers consumed by the hardware to the free list.

16.5.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore, the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the R_BUFF_SIZE register.

The driver (receive BD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the Lbit (1 indicates last buffer in frame), the frame status bits (if L= 1), and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For any other buffer, the length field in the receive BD is written with the default receive buffer length value by the DMA (at the same time the e bit is cleared). For end-of-frame buffers, the receive BD is written with L set and information written to the status bits (M, BC, MC, LG, NO, SH, CR, OV.TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end-of-frame buffer is written with the length of the entire frame, not the length of the last buffer.

For simplicity, the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out-of-specification implementation could result in giant frames. Frames of 2 Kbytes (2048 Kbytes) or larger are truncated by the FEC at 2047 bytes, so software is guaranteed never to see a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the R_DES_ACTIVE register. As frames are received, the FEC fills receive buffers and updates the associated BDs, and then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit equal to 0, it polls this BD once more. If the BD equals 0 a second time, the FEC stops reading receive BDs until the driver writes to R_DES_ACTIVE.

16.5.2 Ethernet Receive Buffer Descriptor (RxBD)

In the RxBD, initialize the E and W bits in the first word and the pointer in the second word. When the buffer has been filled by DMA, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV,

and TR bits and also writes the data length into the first word. If a single receive buffer descriptor is used, the data length is the portion of the data buffer used (the total length of the frame). If the received message spans several data buffers, the data length in all but the last receive buffer (those with the L bit equaling 0) is R_BUFF_SIZE. The data length of the last receive buffer descriptor (the descriptor where the L bit is equal to 1) is the total length of the frame.. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first word of the buffer descriptor are only modified by the Ethernet controller when the I bit is set.

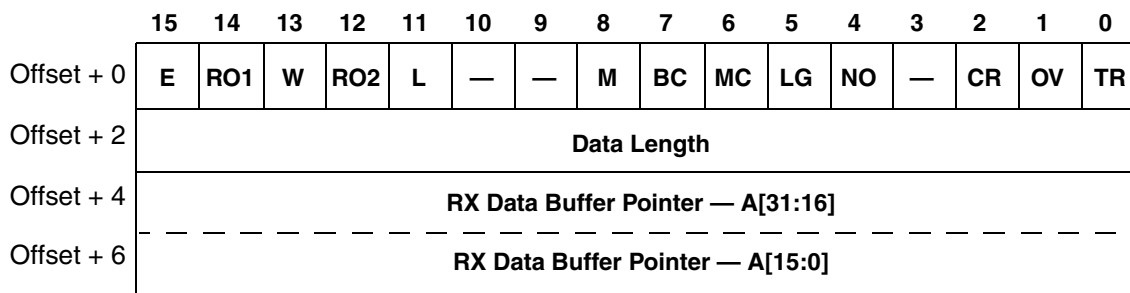


Figure 16-29. RxBD — Receive Buffer Descriptor

The first word of the RXBD contains control and status bits. Its format is detailed in [Table 16-41](#).

Table 16-41. RxBD — Receive Buffer Field Descriptions

Field	Description
E	Empty, written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data or data reception has been aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty or reception is currently in progress.
RO1	Receive software ownership bit This field is reserved for use by software. This read/write bit must not be modified by hardware. Its value does not affect hardware.
W	Wrap, written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in RAM.R_DES_START
RO2	Receive software ownership bit This field is reserved for use by software. This read/write bit must not be modified by hardware. Its value does not affect hardware.
L	Last in frame, written by the FEC. 0 The buffer is not the last in a frame 1 The buffer is the last in a frame
M	Miss, written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a miss by the internal address recognition. Therefore, while in promiscuous mode, you can use the M bit to determine whether the frame was destined to this station. This bit is valid only if the L and the PROM bits are set. 0 The frame was received because of an address recognition hit 1 The frame was received because of promiscuous mode
BC	Set if the DA is broadcast (FF-FF-FF-FF-FF-FF)
MC	Set if the DA is multicast and not bc.

Table 16-41. RxBD — Receive Buffer Field Descriptions (continued)

Field	Description
LG	RX frame length violation, written by the FEC. A frame length greater than R_CNTRL.MAX_FL was recognized. This bit is valid only if the L bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
NO	RX non-octet aligned frame, written by the FEC. A frame that contains a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L bit is set. If this bit is set, the CR bit is not set.
CR	RX CRC error, written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L bit is set.
OV	Overflow, written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits (M, LG, NO, SH, CR, and CL) lose their normal meaning and is zero. This bit is valid only if the L bit is set.
TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set, the frame should be discarded and the other error bits should be ignored because they may be incorrect.
Data Length	Written by the FEC. If a single receive buffer descriptor is being used, the data length is the number of octets written by the FEC into this BD's data buffer (the total length of the frame). If the received message spans several data buffers, the data length is R_BUFF_SIZE in all but the last receive buffer descriptor (those with the L bit equaling 0) . The data length of the last receive buffer descriptor (the descriptor where the L bit is equal to 1) is the total length of the frame in octets.
Rx Buffer Pointer	Written by user. The receive buffer pointer, which always points to the first location of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC.

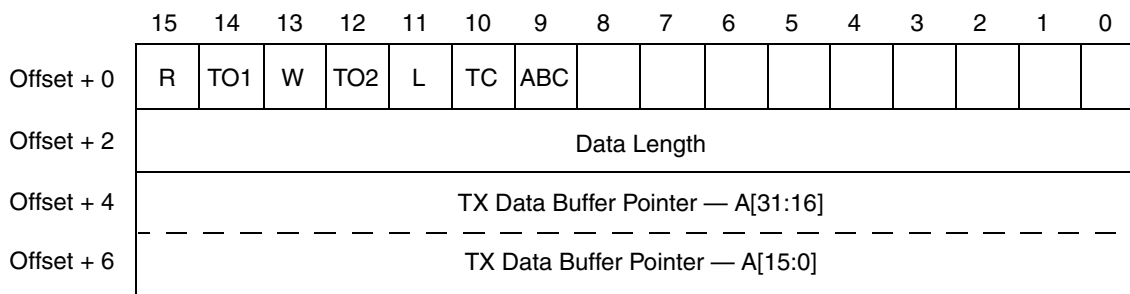
Note: Any time the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to R_DES_ACTIVE.

16.5.3 Ethernet Transmit Buffer Descriptor

Data is presented to the FEC for transmission by arranging the data in buffers referenced by the channel's TxBDS. The Ethernet controller confirms transmission by clearing an ownership bit (R bit) when the buffer's DMA is complete. In the TxBD, initialize the R, W, L, and TC bits and the length (in bytes) in the first word and the buffer pointer in the second word.

The FEC clears the R bit equal to 0 in the first word of the BD when the buffer has been addressed by the direct memory access controllers. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block.

The TxBD fields are detailed in [Figure 16-30](#).

**Figure 16-30. Transmit Buffer Descriptor (TxBD)****NOTE**

The TX data buffer pointer must be evenly divisible by four.

Table 16-42. Transmit Buffer Field Descriptions

Field	Description
R	Ready, written by FEC and user. 0 The data buffer associated with this BD is not ready for transmission. You are free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission, has not been transmitted or is currently being transmitted. No fields of this BD may be modified after this bit is set.
TO1	Transmit software ownership bit. This field is reserved for use by software. This read/write bit must not be modified by hardware. Its value does not affect hardware.
W	Wrap, written by user. This field is reserved for use by software. This read/write bit must not be modified by hardware. Its value does not affect hardware. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in X_DES_START
TO2	Transmit software ownership bit This field is reserved for use by software. This read/write bit must not be modified by hardware. Its value does not affect hardware.
L	Last in frame, written by user. 0 The buffer is not the last in the transmit frame 1 The buffer is the last in the transmit frame
TC	TX CRC, written by user (only valid if L equals 1). 0 End transmission immediately after the last data byte 1 Transmit the CRC sequence after the last data byte
ABC	Append bad CRC, written by user (only valid if L = 1) 0 No effect 1 Transmit an invalid CRC sequence after the last data byte (regardless of TC value)
Data Length	Data length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [21:31] are used by the DMA engine. Bits [16:20] are ignored.

Table 16-42. Transmit Buffer Field Descriptions (continued)

Field	Description
Transmit Data Buffer Pointer	TX buffer pointer, written by user. The transmit buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

Note: After the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the r bit in the first BD for the frame. The driver should follow that with a write to X_DES_ACTIVE, which triggers the FEC to poll the next BD in the ring.

16.6 Network Interface Options

The FEC supports an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The interface mode is selected by the MII_MODE bit in the R_CNTRL register. In MII mode, (R_CNTRL.MII_MODE = 1), there are 18 signals defined by the 802.3 standard and supported by the EMAC. [Table 16-43](#) shows these.

Table 16-43. MII Interface

Signal Description	FEC signals name
Transmit clock	TX_CLK
Transmit enable	TX_EN
Transmit data	TXD[3:0]
Transmit error	TX_ER
Collision	COL
Carrier sense	CRS
Receive clock	RX_CLK
Receive enable	RX_DV
Receive data	RXD[3:0]
Receive error	RX_ER
Management channel clock	MDC
Management channel serial data	MDIO

Serial mode connections to the external transceiver are shown in [Table 16-44](#).

Table 16-44. 7-Wire Interface

Signal Description	FEC signal name
Transmit clock	TX_CLK
Transmit enable	TX_EN
Transmit data	TXD[0]
Collision	COL
Receive clock	RX_CLK

Table 16-44. 7-Wire Interface (continued)

Signal Description	FEC signal name
Receive enable	RX_DV
Receive data	RXD[0]
Unused 860t inputs — tie to GND	RX_ER, CRS, RXD[3:1]
Unused 860t outputs — ignore	TX_ER, TXD[3:1], MDC, MDIO

16.6.1 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ETHER_EN is asserted and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the X_WMRK register), the MAC transmit logic asserts TX_EN and starts transmitting the preamble sequence, the start frame delimiter, and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (carrier sense is asserted). Before transmitting, the controller waits for carrier sense to become inactive, and then determines if carrier sense stays inactive for 60 bit times. If so, then the transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive).

If a collision occurs during transmission of the frame (half-duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit threshold is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so the first 64 bytes do not have to be retrieved again from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (32-bit CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end-of-frame buffer equals 1).

Both buffer (TXB, FEC only) and frame (TFINT, FEC) interrupts may be generated as determined by the settings in the IMASK register.

Transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, XFIFO_UN and XFIFO_ERROR. If the transmit frame length exceeds MAX_FL bytes, the BABT interrupt is asserted; however, the entire frame is transmitted (no truncation).

To pause transmission, set the graceful transmit stop (GTS) bit in the X_CNTRL register. When the GTS is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame finishes or terminates with a collision. After the transmitter has stopped, the GRA interrupt is asserted. If GTS is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit (LSB) first.

16.6.1.1 Duplicate Frame Transmission

The FEC fetches transmit buffer descriptors (TxBDs) and the corresponding transmit data continuously until the transmit FIFO is full. It does not determine whether the TxBD to be fetched is already being processed internally (as a result of a wrap). As the FEC nears the end of the transmission of one frame, it begins to DMA the data for the next frame. To remain one BD ahead of the DMA, it also fetches the TxBD for the next frame. It is possible that the FEC fetches from memory a BD that has already been processed but not yet written back (it is read a second time with the R bit remains set). In this case, the data is fetched and transmitted again.

Using at least three TxBDs fixes this problem for large frames, but not for small frames. To ensure correct operation for large or small frames, one of the following must be true:

- The FEC software driver ensures that there is always at least one TxBD with the ready bit cleared.
- Every frame uses more than one TxBD and every TxBD but the last is written back immediately after the data is fetched.
- The FEC software driver ensures a minimum frame size, n . The minimum number of TxBDs is then $(\text{Tx FIFO Size} \div (n + 4))$ rounded up to the nearest integer (though the result cannot be less than three). The default Tx FIFO size is 192 bytes; this size is programmable.

16.6.2 FEC Frame Reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by asserting `ETHER_EN`, it immediately starts processing receive frames. When `RX_DV` asserts, the receiver checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the frame is processed by the receiver. If a valid PA/SFD is not found, the frame is ignored.

NOTE

The FEC receive block transfers blocks of 16 bytes to the receive buffer even if the message length is not divisible by 16 bytes. Therefore, if the message length is not divisible by 16 bytes, extra bytes are added.

In serial mode, the first 16 bit times of `RX_D0` following assertion of `RX_DV` (`RENA`) are ignored. Following the first 16 bit times, the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first six bytes of the frame have been received, the FEC performs address recognition on the frame.

After a collision window (64 bytes) of data has been received, and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame has been accepted and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to reject the frame. Thus, no collision fragments are presented except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and after the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, SH, CR, OV and TR status bits and the frame length.

Receive buffer (RXB, FEC only) and frame (RFINT, FEC only) interrupts may be generated if enabled by the IMASK register. BABR and RFIFO_ERROR are receive error interrupts. Receive frames are not truncated if they exceed the MAX_FL byte length; however, the BABR interrupt occurs and the LG bit in the receive BD is set.

When the receive frame is complete, the FEC sets the I bit in the receive BD, writes the other frame status bits into the receive BD, and clears the E bit. Next, the Ethernet controller generates a maskable interrupt (RFINT bit in IEVENT, maskable by RFIEN bit in IMASK), indicating a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data LSB first.

16.6.3 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames is illustrated in the following figures.

Address recognition is accomplished through the use of the receive block and ucode running on the descriptor controller. The flowchart shown in [Figure 16-31](#) illustrates the address recognition decisions made by the receive block, while [Figure 16-32](#) illustrates the decisions made by the descriptor controller.

If the DA is a broadcast address and broadcast reject (BC_REJ bit is deasserted), the frame is accepted unconditionally, as shown in [Figure 16-31](#). Otherwise, if the DA is not a broadcast address, the descriptor controller runs the address recognition subroutine, as shown in [Figure 16-32](#).

If the DA is a group (multicast) address and flow control is disabled, the descriptor controller performs a group hash table lookup using the 64-entry hash table programmed in GADDR1 and GADDR2. If a hash match occurs, address recognition hash match bar (AR_HM_B) is set to 0 and the receiver accepts the frame. If flow control is enabled, the descriptor controller does an exact address match check between the DA and the designated pause DA in registers FDXFC_DA1 and FDXFC_DA2. If a pause DA exact match occurs, the address recognition exact match bar (AR_EM_B) is set to 0. If the receive block determines the received frame is a valid pause frame, the frame is rejected. The receiver detects a pause frame with the DA field set to the designated pause DA or the unicast physical address.

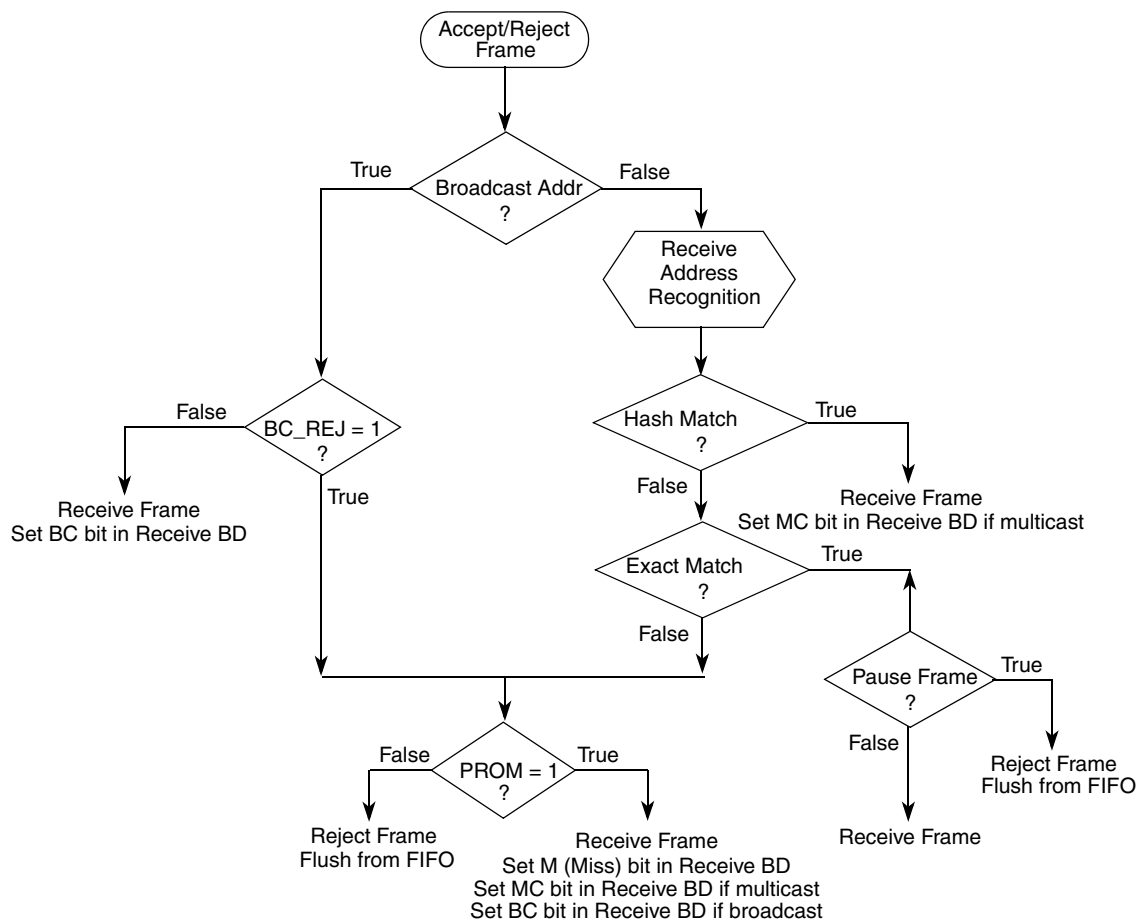
If the DA is the individual (unicast) address, the descriptor controller performs an individual exact match comparison between the DA and 48-bit physical address that you program in the PADDR1 and PADDR2 registers. If an exact match occurs, AR_EM_B is set to 0; otherwise, the descriptor controller does an

individual hash table lookup using the 64-entry hash table programmed in the IADDR1 and IADDR2 registers. In the case of an individual hash match, AR_HM_B is set to 0. Again, the receiver accepts or rejects the frame based on pause frame detection, shown in Figure 16-31.

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then both ar_hm_b and ar_em_b are set to 1. In this case, if promiscuous mode is enabled (r_cntrl.prom = 1), the frame is accepted and the MISS bit in the receive buffer descriptor is set. Otherwise, the frame is rejected and the MISS bit is cleared.

Similarly, if the DA is a broadcast address, broadcast reject (R_CNTRL.BC_REJ) is asserted, and promiscuous mode is enabled, the frame is accepted, and the MISS bit in the receive buffer descriptor is set. Otherwise, the frame is rejected and the MISS bit is cleared.

In general, when a frame is rejected, it is flushed from the FIFO.



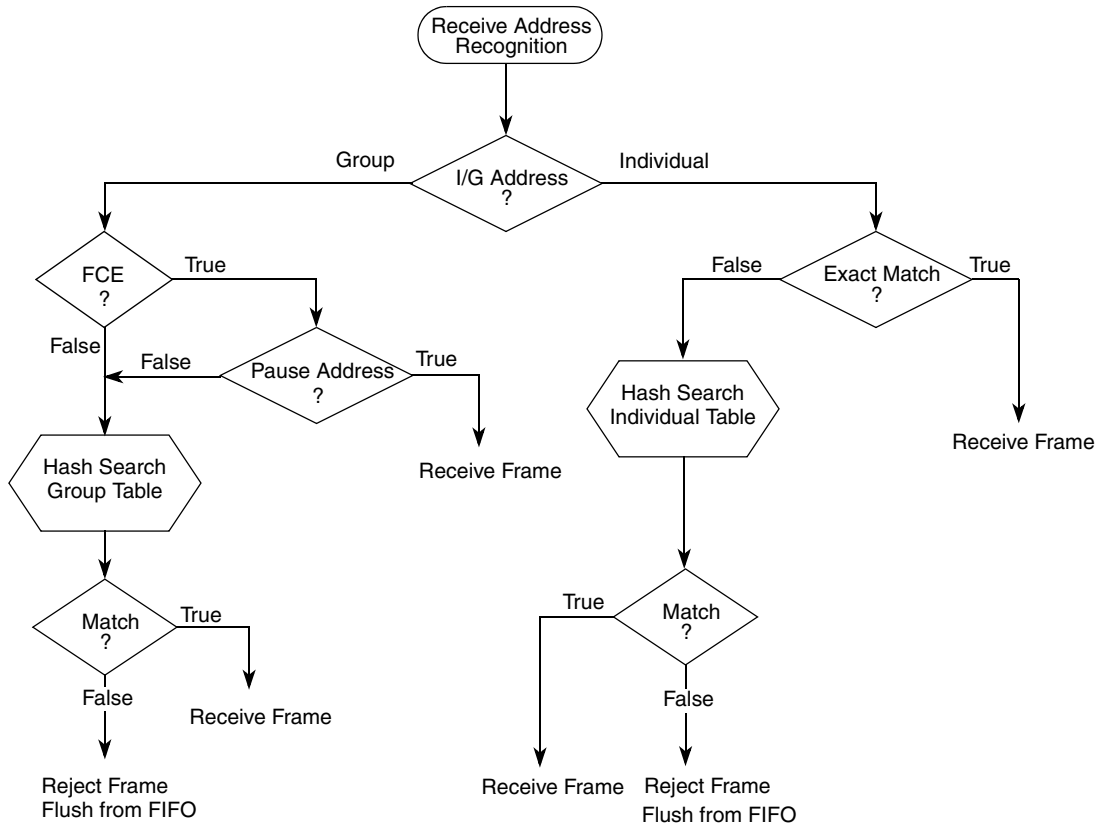
NOTES:

BC_REJ — field in R_CNTRL register (BroadCast REject)

PROM — field in R_CNTRL register (PROMiscuous mode)

Pause Frame — valid PAUSE frame received

Figure 16-31. Ethernet Address Recognition — Receive Block Decisions



NOTES:

FCE — field in R_CNTRL register (Flow Control Enable)

I/G — Individual/Group bit in Destination Address (least significant bit in first byte received in MAC frame)

Figure 16-32. Ethernet Address Recognition — Ucode Decisions

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GADDR1/GADDR2 (group address hash match) or IADDR1/IADDR2 (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GADDR1 (MSB = 1) or GADDR2 (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit set in the hash table, the frame is accepted. Otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The hash table registers must be initialized. The FEC does not support the set group address command, which can be used in the CPM ethernet controllers. You may compute the hash for a particular address in

software or use the set group address command in an off-line CPM channel, retrieve the result and use it to program the FEC hash table registers. The CRC32 polynomial to use in computing the hash is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^4 + X^2 + X + 1 \quad \text{Eqn. 16-1}$$

Table 16-45 includes example destination addresses and corresponding hash values for reference.

Table 16-45. Destination Address to 6-Bit Hash

48-Bit DA	6-Bit Hash (in Hex)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff:ff	0xa	10
8b:ff:ff:ff:ff:ff	0xb	11
0b:ff:ff:ff:ff:ff	0xc	12
3b:ff:ff:ff:ff:ff	0xd	13
7b:ff:ff:ff:ff:ff	0xe	14
5b:ff:ff:ff:ff:ff	0xf	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24
b9:ff:ff:ff:ff:ff	0x19	25
f9:ff:ff:ff:ff:ff	0x1a	26
c9:ff:ff:ff:ff:ff	0x1b	27
59:ff:ff:ff:ff:ff	0x1c	28
79:ff:ff:ff:ff:ff	0x1d	29

Table 16-45. Destination Address to 6-Bit Hash (continued)

48-Bit DA	6-Bit Hash (in Hex)	Hash Decimal Value
29:ff:ff:ff:ff:ff	0x1e	30
19:ff:ff:ff:ff:ff	0x1f	31
d1:ff:ff:ff:ff:ff	0x20	32
f1:ff:ff:ff:ff:ff	0x21	33
b1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7f:ff:ff:ff:ff:ff	0x28	40
4f:ff:ff:ff:ff:ff	0x29	41
1f:ff:ff:ff:ff:ff	0x2a	42
3f:ff:ff:ff:ff:ff	0x2b	43
bf:ff:ff:ff:ff:ff	0x2c	44
9f:ff:ff:ff:ff:ff	0x2d	45
df:ff:ff:ff:ff:ff	0x2e	46
ef:ff:ff:ff:ff:ff	0x2f	47
93:ff:ff:ff:ff:ff	0x30	48
b3:ff:ff:ff:ff:ff	0x31	49
f3:ff:ff:ff:ff:ff	0x32	50
d3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3d:ff:ff:ff:ff:ff	0x38	56
0d:ff:ff:ff:ff:ff	0x39	57
5d:ff:ff:ff:ff:ff	0x3a	58
7d:ff:ff:ff:ff:ff	0x3b	59
fd:ff:ff:ff:ff:ff	0x3c	60
dd:ff:ff:ff:ff:ff	0x3d	61
9d:ff:ff:ff:ff:ff	0x3e	62
bd:ff:ff:ff:ff:ff	0x3f	63

16.6.4 Full-Duplex Flow Control

Full-duplex flow control allows you to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (X_CNTRL.FDEN asserted) and flow control enable (R_CNTRL.FCE) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in [Table 16-46](#). In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 16-46. PAUSE Frame Field Specification

48-Bit Destination Address	0180_c200_0001 or Physical Address
48-bit source address	any
16-bit type	8808
16-bit opcode	0001
16-bit pause duration	0000 to ffff

Pause frame detection is performed by the receiver and descriptor controller modules. The descriptor controller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, graceful transmit stop is asserted by the FEC internally. When transmission has paused, the graceful stop complete (GRA) interrupt is asserted and the pause timer begins to increment. The pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until pause_duration slot times have expired. When pause_duration expires, graceful transmit stop is deasserted, allowing MAC data frame transmission to resume. The receive flow control pause (RFC_PAUSE) status bit is asserted while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and software must assert flow control pause (TFC_PAUSE). On assertion of TFC_PAUSE, the transmitter asserts graceful transmit stop internally. When the transmission of data frames stops, the graceful stop complete (GRA) interrupt asserts. Following GRA assertion, the pause frame is transmitted. When pause frame transmission is complete, TFC_PAUSE and graceful transmit stop are deasserted internally.

During pause frame transmission, the transmit hardware places data into the transmit data stream from the registers shown in [Table 16-47](#).

Table 16-47. Transmit Pause Frame Registers

PAUSE Frame Fields	FEC Register	Register Contents
48-bit destination address	{FDXFC_DA1[0:31], fDXFC_DA2[0:15]}	0180_c200_0001
48-bit source address	{PADDR1[0:31], PADDR2[0:15]}	Physical address
16-bit type	PADDR2[16:31]	8808
16-bit opcode	OP_PAUSE[0:15]	0001
16-bit pause duration	OP_PAUSE[16:31]	0000 to ffff

Specify the desired pause duration in the OP_PAUSE register.

When the transmitter is paused due to receiver/descriptor controller pause frame detection, transmit flow control pause (TFC_PAUSE) may continue to be asserted and cause the transmission of a single pause frame. In this case, the GRA interrupt is not asserted.

16.6.5 Inter-Packet Gap Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit counts, the following frame may be discarded by the receiver.

16.6.6 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a jam pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the jam pattern is sent after the end of the preamble sequence.

If a collision occurs within 64 byte times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 64 byte times, no retransmission is performed and the end-of-frame buffer is closed with an LC error indication.

16.6.7 Internal and External Loopback

Internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Internal and external loopback are configured using combinations of the LOOP and DRT bits in the R_CNTRL register and the FDEN bit in the X_CNTRL register.

For internal and external loopback, set FDEN equal to 1.

For internal loopback, set LOOP equal to 1 and DRT equal to 0. TX_EN and TX_ER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the internal system clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being direct memory addressed to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback, set LOOP equal to 0 and DRT equal 0, and then configure the external transceiver for loopback.

16.6.8 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC receive BDs, the IEVENT register and the MIB block counters.

16.6.9 Transmission Errors

- Transmitter Underrun
 - If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the X_STATUS register. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.
 - The XFIFO_UN interrupt is asserted if enabled in the IMASK register
- Carrier Sense Lost During Frame Transmission
 - When this error occurs and no collision is detected in the frame, the FEC sets the CSL bit in X_STATUS register. The frame is transmitted normally. No retries are performed as a result of this error.
 - No interrupt is generated as a result of this error
- Retransmission Attempts Limit Expired
 - When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are then flushed and closed, and the RL bit is set in the X_STATUS register. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.
 - The COL_RETRY_LIM interrupt is asserted if enabled in the IMASK register
- Late Collision
 - When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are then flushed and closed, and the LC bit is set in the X_STATUS register. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.
 - The LATE_COL interrupt is asserted if enabled in the IMASK register
- Heartbeat
 - Some transceivers have a self-test feature called heartbeat or signal quality error. To signify a good self-test, the transceiver indicates a collision to the FEC within 20 clocks after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver seems to be functioning properly. This is called the heartbeat condition.
 - If the HBC bit is set in the X_CNTRL register and the heartbeat condition is not detected by the FEC after a frame transmission, a heartbeat error occurs. When this happens, the FEC closes the buffer, sets the HB bit in the X_STATUS register, and generates the HBERR interrupt if it is enabled.

16.6.10 Reception Errors

- **Overflow Error**
 - If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the receive status word. All subsequent data in the frame is discarded and subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space becomes available. At this point, the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.
- **Non-Octet Error (Dribbling Bits)**
 - The Ethernet controller manages up to seven dribbling bits when the receive frame terminates non-octet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, the frame non-octet aligned (no) error is reported in the receive BD. If there is no CRC error, no error is reported.
- **CRC Error**
 - When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.
- **Frame Length Violation**
 - When the receive frame length exceeds MAX_FL bytes, the BABT interrupt is generated and the LG bit in the end-of-frame receive BD is set. The frame is not truncated (truncation occurs if the frame length exceeds 2047 bytes).
- **Truncation**
 - When the receive frame length exceeds 2047 bytes, the frame is truncated and the TR bit is set in the receive BD.

Chapter 17

General Purpose Timers (GPT)

17.1 Introduction

Eight general-purpose timer (GPT) pins are configurable for:

- Input capture
- Output compare
- Pulse width modulation (PWM) output
- Simple GPIO
- Internal CPU timer

The General Purpose Timer provides 8 independent Timer Channels that perform General Purpose I/O, Input Capture, Output Compare, Pulse Width Modulation and Internal CPU timer functions. An external I/O pin is associated with each Timer Channel. A separate 16-bit prescaler and 16-bit counter is associated with each timer channel, thus achieving a range of 32-bits (but only 16-bit resolution).

17.1.1 Modes of Operation

- **Input Capture**—In this mode the I/O pin is an Input. There are two counters used for each timer channel in this mode. The first counter is the Internal counter and the second is the Updown counter. After enabled, when specified capture event occurs (rising edge, falling edge, either edge, or pulse - two consecutive edges), the internal counter value is latched in the status register. If enabled, a CPU interrupt is generated. The Input Capture function has the following submodes, which are controlled by the ICM bits in section [Section 17.2.2, “Register Descriptions”](#) GPT Enable and Mode Select Register:
 - Normal input capture mode. Only internal counter is active.
 - Up submode as well as normal input capture submode. Both updown counter and internal counter are active.
 - Updown submode as well as input capture submode. Both counters are active. A pair of timer channels are used to implement this mode.
 - Rotary counter as well as input capture counter mode. Both counters are active. A pair of timer channels are used to implement this mode.
 - When changing from one submode into another submode, the TIMER should be disabled first.
- **Output Compare**—In this mode the I/O pin is an Output. When enabled, the counters run until they reach the programmed Terminal Count value. At this point, the specified output event is generated (toggle, pulse high, or pulse low). If enabled, a CPU interrupt is generated.

- **PWM**—In this mode the I/O pin is an Output. The user can program period and width values to create an adjustable, repeating output waveform on the I/O pin. A CPU interrupt can be generated at the beginning of each PWM Period, at which time a new Width value can be loaded. The new Width value, which represents ON time, is automatically applied at the beginning of the next period. There is no interrupt at the beginning of the first PWM Period. This mode is suitable for PWM audio encoding.
- **Simple GPIO**—In this mode the I/O pin operates as a GPIO pin. It can be specified as Input or Output, according to the programmable GPIO field. GPIO mode is mutually exclusive of Input Capture, Output Compare and PWM modes.
- **CPU Timer**—The I/O pin is not used in this mode. After enabled, the counters run until they reach a programmed Terminal Count. When this occurs, an interrupt can be generated to the CPU. This Timer mode can be used simultaneously with the Simple GPIO mode.

17.1.2 Detailed Signal Descriptions

Table 17-1 provides detailed descriptions of the external GPTimer signals.

Table 17-1. GPIO External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
GPTimer[0:7]	I/O	GPTimer 0—7. Each pin can be individually set to act as input or output, according to application needs.	
		State Meaning	Asserted/Negated—Defined per application.
		Timing	Assertion/Negation —Inputs can be asserted completely asynchronously. Outputs are asynchronous to any externally visible clock

17.2 Memory Map and Register Definition

Each GPT Timer Channel uses 4 32-bit registers. These registers are located at an offset from IMMRBAR of 0x0B00. Register addresses are relative to this offset. Therefore, the actual register address is: $\text{IMMRBAR} + 0x0B00 + \text{Register Offset} = \text{register address}$.

17.2.1 Memory Map

Table 17-2 shows the memory map for the local access registers:

Table 17-2. GPTimer Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x0B00	GPT0 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B10	GPT1 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B20	GPT2 Enable and Mode Select Register	R/W	17.2.2.1/17-4

Table 17-2. GPTimer Memory Map (continued)

Offset or Address	Register	Access	Section/Page
0x0B30	GPT3 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B40	GPT4 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B50	GPT5 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B60	GPT6 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B70	GPT7 Enable and Mode Select Register	R/W	17.2.2.1/17-4
0x0B04	GPT0 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B14	GPT1 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B24	GPT2 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B34	GPT3 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B44	GPT4 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B54	GPT5 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B64	GPT6 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B74	GPT7 Counter Input and Updown Counter Output Register	R/W	17.2.2.2/17-8
0x0B08	GPT0 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B18	GPT1 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B28	GPT2 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B38	GPT3 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B48	GPT4 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B58	GPT5 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B68	GPT6 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B78	GPT7 PWM Configuration Register	R/W	17.2.2.3/17-9
0x0B0C	GPT0 Status Register	R	17.2.2.4/17-10
0x0B1C	GPT1 Status Register	R	17.2.2.4/17-10
0x0B2C	GPT2 Status Register	R	17.2.2.4/17-10
0x0B3C	GPT3 Status Register	R	17.2.2.4/17-10
0x0B4C	GPT4 Status Register	R	17.2.2.4/17-10
0x0B5C	GPT5 Status Register	R	17.2.2.4/17-10
0x0B6C	GPT6 Status Register	R	17.2.2.4/17-10
0x0B7C	GPT7 Status Register	R	17.2.2.4/17-10

17.2.2 Register Descriptions

17.2.2.1 GPT0 – GPT7 Enable and Mode Select Registers

Register address offset from IMMRBAR of 0x0B00:

0x0B00, 0x0B10, 0x0B20, 0x0B30, 0x0B40, 0x0B50, 0x0B60, 0x0B70

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OCPW										OCT		ICM		ICT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				CE		Stop_Cont	Open_Drn	IntEn			GPIO			Timer_MS		
W	S ¹															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 17-1. GPT0–GPT7 Enable and Mode Select Registers
(The register is repeated for reference.)

Table 17-3. GPT0–GPT7 Enable and Mode Select Field Descriptions (Sheet 1 of 4)

Field	Description
OCPW	Output Compare Pulse Width. Applies to OC Pulse types only. This field specifies the number of IP bus clocks (non-prescaled) to create a short output pulse at each Output Event. This pulse is generated at the end of the OC period and overlays the next OC period (rather than adding to the period).
OCT	<p>Output Compare Type. Describes action to occur at each output compare event, as follows:</p> <ul style="list-style-type: none"> 00 Special case, output is immediately forced low without respect to each output compare event. 01 Output pulse highs, initial value is low (OCPW field applies). 10 Output pulses low, initial value is high (OCPW field applies). 11 Output toggles. <p>GPIO modalities can be used to achieve an initial output state prior to enabling OC mode. It is important to move directly from one GPIO output mode to another OC mode and not to pass through the Timer_MS=000 state.</p> <p>To prevent the Internal Timer Mode from engaging during the GPIO state, CE bit should be held low during the configuration steps.</p> <p>GPIO initialization is needed when presetting a Timer I/O to 1 in conjunction with a simple toggle OCT setting.</p> <p>Note: For Stop Mode operation (see Stop_Cont bit below) it is necessary to pass through the mode_sel = 0 state to restart the output compare counters with their programmed values. See prescale and count fields in 17.1.1/17-1.</p>

Register address offset from IMMRBAR of 0x0B00:

0x0B00, 0x0B10, 0x0B20, 0x0B30, 0x0B40, 0x0B50, 0x0B60, 0x0B70

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OCPW										OCT		ICM		ICT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				CE		Stop_Cont	Open_Drn	IntEn			GPIO			Timer_MS		
W	S ¹															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 17-1. GPT0–GPT7 Enable and Mode Select Registers
(The register is repeated for reference.)

Table 17-3. GPT0–GPT7 Enable and Mode Select Field Descriptions (Sheet 2 of 4)

Field	Description
ICM	<p>Input Capture Mode—describes the input capture sub-mode as follows:</p> <ul style="list-style-type: none"> 00 Normal input capture submode. 01 Up submode as well as normal input capture submode. Timer updown counter increases one if it detects IC event. 10 Updown submode as well as input capture submode. Timers must be used in pairs. Timer 0 is paired with Timer 1, Timer 2 is paired with Timer 3, Timer 4 is paired with Timer 5 and Timer 6 is paired with Timer 7. For example, the timer0 updown counter increases by one if timer0 detects IC event. The Timer 0 Updown Counter decreases by one if timer1 detects an IC event. The timer 0 Updown Counter remains unchanged if an IC event occurs on both channels during a single prescaled clock count. Timers 2 and 3, Timers 4 and 5, and Timers 6 and 7 operate in a similar fashion. 11 Rotary submode as well as input capture submode. When an IC event is detected on Timer 0, the Timer 0 Updown Counter is incremented by 1 if Timer 1 is driven to a logic 0. the Timer 0 Updown Counter decrements by 1 if Timer 1 is driven to a logic 1. Timers 2 and 3, Timers 4 and 5 and Timers 6 and 7 operate in a similar fashion. When updown counter overflow or underflow occurs, a CPU interrupt can be generated if the interrupt enable bit is set. <p>The IC event type is defined by the ICT field.</p> <p>Note: The updown counter value can be read from 17.2.2.2/17-8 bits 16-31. The value represents how many times an event happens. It is independent of the IC counter, which runs when enabled and latches the value when the IC event happens. Normally counter means IC counter (input capture counter) when timer is in Input Capture mode. When changing from one submode into another submode, the TIMER should be disabled.</p>
ICT	<p>Input Capture Type. Describes the input transition type required to trigger an input capture event, as follows:</p> <ul style="list-style-type: none"> 00 Any input transition causes an IC event. 01 IC event occurs at input rising edge. 10 IC event occurs at input falling edge. 11 IC event occurs at any input pulse (i.e., at 2nd input edge). <p>BE AWARE: For ICT 11 (pulse capture), status register records only the pulse width.</p>

General Purpose Timers (GPT)

Register address offset from IMMRBAR of 0x0B00:

0x0B00, 0x0B10, 0x0B20, 0x0B30, 0x0B40, 0x0B50, 0x0B60, 0x0B70

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OCPW										OCT		ICM		ICT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				CE		Stop_Cont	Open_Drn	IntEn			GPIO			Timer_MS		
W	S ¹															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 17-1. GPT0–GPT7 Enable and Mode Select Registers
(The register is repeated for reference.)

Table 17-3. GPT0–GPT7 Enable and Mode Select Field Descriptions (Sheet 3 of 4)

Field	Description
CE	<p>Counter Enable. Bit enables or resets the internal counter during Internal timer modes only. CE must be high to enable these modes. If low, counter is held in reset.</p> <p>This bit is secondary to the timer mode select bits (Timer_MS). If the Timer_MS contains 1XX, the internal timer modes are enabled. CE can then enable or reset the internal counter without changing the Timer_MS field.</p> <p>GPIO operation is also available in this mode. 1 = enabled</p>
STOP_CONT	<p>Stop Continuous—Applies to multiple modes, as follows:</p> <ul style="list-style-type: none"> 0 Stop 1 Continuous <p>IC mode</p> <ul style="list-style-type: none"> • Stop operation—At each IC event, counter is reset. • Continuous operation—counter is not reset at each IC event. • Effect is to create Status count values that are cumulative between Capture events. If the Pulse Mode Capture mode is specified, the Stop_Cont bit is not used, operation fixed as if it were Stop. <p>OC mode</p> <ul style="list-style-type: none"> • Stop operation—Counter resets and stops at first OC event. <p>Note: Software needs to pass through Timer_MS=000 state to restart timer.</p> <ul style="list-style-type: none"> • Continuous operation—counter resets and continues at each OC event. • Effect is to create back-to-back periodic OC events. <p>PWM mode</p> <ul style="list-style-type: none"> • Bit not used, operation is always Continuous. <p>CPU Timer mode</p> <ul style="list-style-type: none"> • Stop operation—On counter expiration, Timer waits until Status bit is cleared by passing through Timer_MS=000 state before beginning a new cycle. • Continuous operation—On counter expiration, Timer resets and immediately begin a new cycle. • Effect is to generate fixed periodic timeouts. <p>GPIO modes</p> <ul style="list-style-type: none"> • Bit not used.

Register address offset from IMMRBAR of 0x0B00:

0x0B00, 0x0B10, 0x0B20, 0x0B30, 0x0B40, 0x0B50, 0x0B60, 0x0B70

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OCPW										OCT		ICM		ICT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R						Stop_Cont	Open_Drn	IntEn			GPIO			Timer_MS		
W	S ¹			CE												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 17-1. GPT0–GPT7 Enable and Mode Select Registers
(The register is repeated for reference.)

Table 17-3. GPT0–GPT7 Enable and Mode Select Field Descriptions (Sheet 4 of 4)

Field	Description
OPEN-DRN	Open Drain 0 Normal I/O 1 Open Drain emulation—affects all modes that drive the I/O pin (GPIO, OC, & PWM). Any output 1 is converted to a tri-state condition on the I/O pin.
INTEN	Enable interrupt—enables interrupt generation to the CPU for all modes (IC, OC, PWM, and Internal Timer).
GPIO	GPIO mode type. Simple GPIO functionality that can be used simultaneously with the Internal Timer mode. It is not compatible with IC, OC, or PWM modes, since these modes require the usage of the I/O pin. 0x Timer enabled as simple GPIO input 10 Timer enabled as simple GPIO output, value=0 11 Timer enabled as simple GPIO output, value=1 (tri-state if Open_Drn=1) While in GPIO modes, internal timer mode is also available. To prevent undesired timer expiration, set the CE bit low.
TIMER_MS	Timer Mode Select (and module enable). 000 Timer module not enabled. Associated I/O pin is in input state. All Timer operation is completely disabled. Control and status registers remain accessible. This mode should be entered when timer is to be re-configured, except where the user does not want the I/O pin to become an input. 001 Timer enabled for input capture. Sub-mode can be set in field ICM. 010 Timer enabled for output compare. 011 Timer enabled for PWM. 1xx timer enabled for simple GPIO. Internal timer modes available. CE bit controls timer counter.

¹ Special- This bit must be set as 0 in normal working mode.

17.2.2.2 GPT0 – GPT7 Input and Up/Down Counter Output Register

Register address offset from IMMRBAR of 0x0B00:

0x0B04, 0x0B14, 0x0B24, 0x0B34, 0x0B44, 0x0B54, 0x0B64, 0x0B74

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Prescale															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Count															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-2. GPT0–GPT7 Input and Up/Down Counter Output Registers
(The register is repeated for reference.)

Table 17-4. GPT0–GPT7 Counter Input and Updown Counter Output Field Descriptions

Field	Description
PRESCALE	Prescale amount applied to internal counter (in IP bus clocks). Note: The prescale field should be written prior to enabling any timer mode. A prescale of 0x0001 means one IP bus clock per count increment. If prescale is 0 when any timer mode is started, it results in an effective prescale of 64K. The counter immediately begins and an output event occurs with the 64K prescale, rather than the desired value.
COUNT	Input Sets number of prescaled counts applied to reference events, as follows: OC—Number of prescaled counts counted before creating output event. PWM—Number of prescaled counts defining the PWM output period. Internal Timer—Number of prescaled counts counted before timer expires. Note: Reading this register only returns the programmed value, intermediate values of the internal counter are not available to software. Output IC—When ICM is equal to 01/10/11, reading this field returns the internal updown counter value. Note: Internal updown counter starts at 0. Writing this field has no effect.

17.2.2.3 GPT0 – GPT7 PWM Configuration Register

Register address offset from IMMRBAR of 0x0B00:

0x0B08, 0x0B18, 0x0B28, 0x0B38, 0x0B48, 0x0B58, 0x0B68, 0x0B78

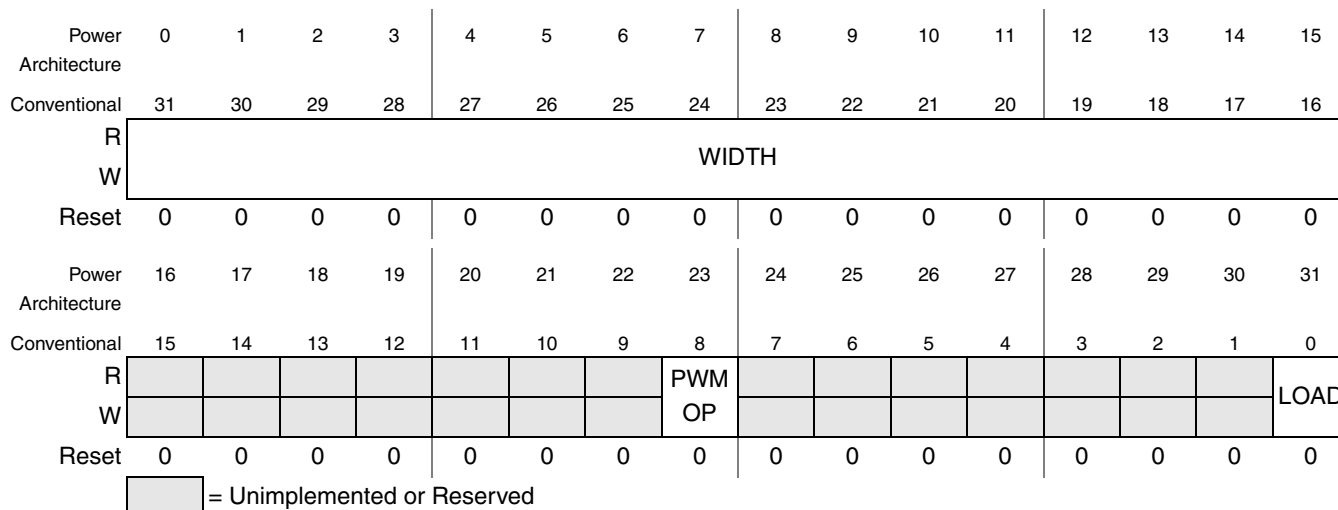


Figure 17-3. GPT0–GPT7 PWM Configuration Registers)

Table 17-5. GPT0–GPT7 PWM Configuration Field Descriptions

Field	Description
WIDTH	PWM only. Defines ON time for output in prescaled counts. The PWM period is determined by the GPT Counter Input and Updown Counter Output Register. ON time overlays the period time. If WIDTH = 0, output is always OFF. If WIDTH exceeds count value, output is always ON. ON and OFF polarity is set by the PWMOP bit.
PWMOP	Pulse Width Mode Output Polarity. Defines PWM output polarity for OFF time. Opposite state is ON time polarity. PWM cycles begin with ON time. 0 = OFF TIME is a logic 0. 1 = OFF TIME is a logic 1.
LOAD	Bit forces immediate period update. Bit auto clears itself. A new period begins immediately with the current count and width settings. If LOAD = 0, new count or width settings are not updated until end of current period. If LOAD = 1, new Count and Width settings take effect immediately. Note: Prescale setting is not part of this process. Changing prescale value while PWM is active causes unpredictable results for the period in which it was changed. The same is true for PWMOP bit.

17.2.2.4 GPT0 – GPT7 Status Register

This is a read-only register

Register address offset from IMMRBAR of 0x0B00:

0x0B0C, 0x0B1C, 0x0B2C, 0x0B3C, 0x0B4C, 0x0B5C, 0x0B6C, 0x0B7C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CAPTURE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		OVF						PIN				UDOV	TEXP	PWMP	COMP	CAFT
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 17-4. GPT0–GPT7 PWM Configuration Registers
(The register is repeated for reference.)

Table 17-6. GPT0–GPT7 PWM Configuration Field Descriptions

Field	Description
CAPTURE	Read of internal counter, latch at reference event. This is pertinent only in IC mode, in which case it represents the count value at the time the Input Event occurred. Capture status does not shadow the internal counter while an event is pending, it is updated only at the time the Input Event occurs. Note: If ICT is set to 11, which is Pulse Capture Mode, the Capture value records the width of the pulse. Also, the Stop_Count bit is irrelevant in Pulse Capture Mode, operation is as if Stop_Count were 0.
OVF	Represents how many times internal counter has rolled over. This is pertinent only during IC mode and would represent an extremely long period of time between Input Events. However, if Stop_Count = 1 (indicating cumulative reporting of Input Events), this field could come into play. Note: This field is cleared by any sticky bit status write in the 5 bit fields below (27,28, 29, 30, 31).
PIN	Registered state of the I/O PIN (all modes). The IP Bus Clock registers the state of the I/O input. Valid, even if Timer is not enabled.
UDOV	Updown counter have wrapped in up/updown/rotary IC submode, i.e. overflowed from 0xFFFF to 0x0000 or underflowed from 0x0000 to 0xFFFF. Cleared by writing 1 to this bit position. Also cleared if Timer_MS is 000 (i.e., Timer not enabled). See Note.
TEXP	Timer Expired in Internal Timer mode. Cleared by writing 1 to this bit position. Also cleared if Timer_MS is 000 (i.e., Timer not enabled). See Note.
PWMP	PWM end of period occurred. Cleared by writing 1 to this bit position. Also cleared if Timer_MS is 000 (i.e., Timer not enabled). See Note.
COMP	OC reference event occurred. Cleared by writing 1 to this bit position. Also cleared if Timer_MS is 000 (i.e., Timer not enabled). See Note.
CAFT	IC reference event occurred. Cleared by writing 1 to this bit position. Also cleared if Timer_MS is 000 (i.e., Timer not enabled). See Note.

Register address offset from IMMRBAR of 0x0B00:

0x0B0C, 0x0B1C, 0x0B2C, 0x0B3C, 0x0B4C, 0x0B5C, 0x0B6C, 0x0B7C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CAPTURE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		OVF						PIN				UDOV	TEXP	PWMP	COMP	CAFT
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 17-4. GPT0–GPT7 PWM Configuration Registers
(The register is repeated for reference.)

Table 17-6. GPT0–GPT7 PWM Configuration Field Descriptions (continued)

Field	Description
-------	-------------

Note: To clear any of these bits, it is necessary to clear **all** of them. An 1F must be written to bits 27:31.

17.3 Functional Description

The General Purpose Timer provides 8 independent Timer Channels that perform General Purpose I/O, Input Capture, Output Compare, Pulse Width Modulation and Internal CPU timer functions. An external I/O pin is associated with each Timer Channel. A separate 16-bit prescaler and 16-bit Internal Counter is associated with each timer channel, thus achieving a range of 32-bits (but only 16-bit resolution). The 16-bit Internal Counter is not visible to the user. Thus, its value cannot be modified or read directly by software. In general, this 16-bit Internal Counter is used in one of two basic methods. The value of this counter can be captured upon the occurrence of some event. Then, the contents of the Internal Counter can be captured again at a second event. The difference between the two values is the number of prescaled clock counts between the two events. The second methodology involves adding a value to the Internal Counter and putting this value into a compare register. When the counter increments to this calculated value, a predetermined event can be programmed to occur.

An UP-DOWN counter is also implemented. This counter is visible to the user. In general, this counter increments in response to certain events and decrement in response to other events. The use of this counter is discussed below.

17.3.1 Input Capture Mode

In this mode the Timer I/O pin is an Input. There are two counters used for each timer channel in this mode. The first counter is the Internal counter, and the second counter is the Updown counter. After enabled, when the specified capture event occurs (rising edge, falling edge, either edge, or pulse—two consecutive edges), the Internal Counter value is latched in the status register. If enabled, a CPU interrupt is generated. The input capture function has the following submodes, which are controlled by the ICM bits in section [Section 17.2.2, “Register Descriptions”](#): normal input capture mode, up mode, up down mode, and rotary mode.

17.3.1.1 Normal Input Capture Mode (IC MODE)

Only the Internal Counter is active in this mode. The IC Mode is selected by setting the ICM field of the GPT Enable and Mode Select Register associated with a particular timer channel to 00. In this mode, the ICT bits of the GPT Enable and Mode Select Register is used to configure the Timer Channel pin as an input and to respond to any transition, a positive transition, a negative transition, or a pulse consisting of two consecutive edges. In the cases of any transition, positive transition or negative transition, the value of the Internal Counter is latched into the CAPTURE field of the GPT Status Register associated with the particular timer channel. If enabled by the IntEn bit of the GPT Enable and Mode Select Register for a particular channel, an interrupt to the CPU is generated.

17.3.1.2 UP Submodule

When a Timer Channel is programmed to use the UP Mode, both the Updown Counter and Internal Counter are active. The Updown counter is incremented by one each time an Input Capture Event, as defined by the ICT field of the GPT Enable and Mode Select Register, occurs. The value of the Updown counter can be obtained by reading the COUNT field of the GPT Counter Input and Updown Counter Output Register for the particular timer channel.

An interrupt is generated, if enabled, when the UP Down Counter overflows.

The UP DOWN Counter cannot be modified by software.

17.3.1.3 UP DOWN Mode

When using the UP DOWN submode, the Updown Counter and Internal Counter are active. A pair of GPT channels must be used to implement this mode. Timer Channel 0 is paired with Timer Channel 1, Timer Channel 2 is paired with Timer Channel 3, Timer Channel 4 is paired with Timer Channel 5 and Timer Channel 6 is paired with Timer Channel 7. The ICM field of the GPT Enable and Mode Select Register for both timer channels must be set to 10.

The ICT field in the GPT Enable and Mode Select Register for both channels must be programmed to detect the desired transitions. After a pair of channels is properly programmed, the Updown Counter of one channel increments each time an Input Capture Event occurs on this channel of the pair and decrements by 1 when an Input Capture Event occurs on the other channel of the pair. That is, for the pair of channels consisting of Timer Channel 0 and Timer Channel 1, the Updown counter associated with Timer Channel 0 increments if an Input Capture event occurs on Timer Channel 0 and decrements if an Input Capture event occurs on Timer Channel 1.

An interrupt is generated, if enabled, when the Updown Counter either underflows or overflows.

17.3.1.4 Rotary Mode

When using the ROTARY submode, both the Updown Counter and Internal Counter are active. A pair of GPT channels must be used to implement this mode. Timer Channel 0 is paired with Timer Channel 1, Timer Channel 2 is paired with Timer Channel 3, Timer Channel 4 is paired with Timer Channel 5 and Timer Channel 6 is paired with Timer Channel 7. The ICM field of the GPT Enable and Mode Select Register for both timer channels must be set to 11.

The ICT field in the GPT Enable and Mode Select Register for one channel of the pair must be programmed to detect the desired transitions. After a pair of channels is properly programmed, the Updown Counter increments each time an Input Capture Event occurs on one channel of the pair if the logic level on the input of the other Channel of the pair is a logic 0. The Updown counter decrements each time an Input Capture Event occurs on one channel of the pair if the logic level on the input of the other Channel of the pair is a logic 1.

For instance, if Timer Channel 0 is programmed to recognize positive transitions, the Updown counter associated with Timer Channel 0 increments each time a positive transition is detected on Timer Channel 0 if the logic level on Timer Channel 1 is a logic 0. The Updown counter associated with Timer Channel 0 decrements each time a positive transition is detected on Timer Channel 0 if the logic level on Timer Channel 1 is a logic 1.

17.3.2 Changing Sub-Modes

When using any mode of the GPT, unpredictable results can occur by arbitrarily switching from one mode to another mode. Before re-configuring a timer channel to a different mode of operation, the Timer Channel should be disabled by setting the `TIMER_MS` field in the GPT Enable and Mode Select Register to 000.

17.3.3 Output Compare

In this mode the I/O pin is an Output. When enabled the counters run until they reach the programmed Terminal Count value. At this point, the specified output event is generated (toggle, pulse hi, or pulse low). If enabled, a CPU interrupt is generated.

17.3.4 Force Output Low Immediately

The `OCT` field of the GPT Enable and Mode Select Register for a particular Timer Channel is written to 00 to force its associated Timer Channel Pin to a logic 0. No Output Compare event has to occur to force the Timer pin to a logic 0.

17.3.5 Output Pulse High

A GPT Timer Channel can be programmed to issue a single pulse with positive polarity in response to an Output Compare Event. The `OCT` field and the `TIMER_MS` field of the GPT Enable and Mode Select Register must respectively be programmed to 01 and 010 to enable this mode. The High Time of the pulse, specified in non-prescaled IP Bus clocks, is programmed into the `OCPW` field of the GPT Enable and Mode Select Register. To create an Output Compare Event, a value can be written to the `COUNT` field of the GPT Counter Input and Updown Counter Output Register that specifies the time, expressed in prescaled IP Bus Clocks, when the next Output Compare event occurs. For example, if the `OCPW` field is written to 3, the `PRESCALE` field is written to 2 and the Count field is written to 4 the result is a positive pulse which is 3 IP Bus clocks wide that occurs eight (Prescale = 2, Count = 4) IP Bus clocks after writing the `TIMER_MS` field of the GPT Enable and Mode Select Register to 010. If the `STOP_CONT` bit is set to 1 - continuous operation, the following wave form is generated.

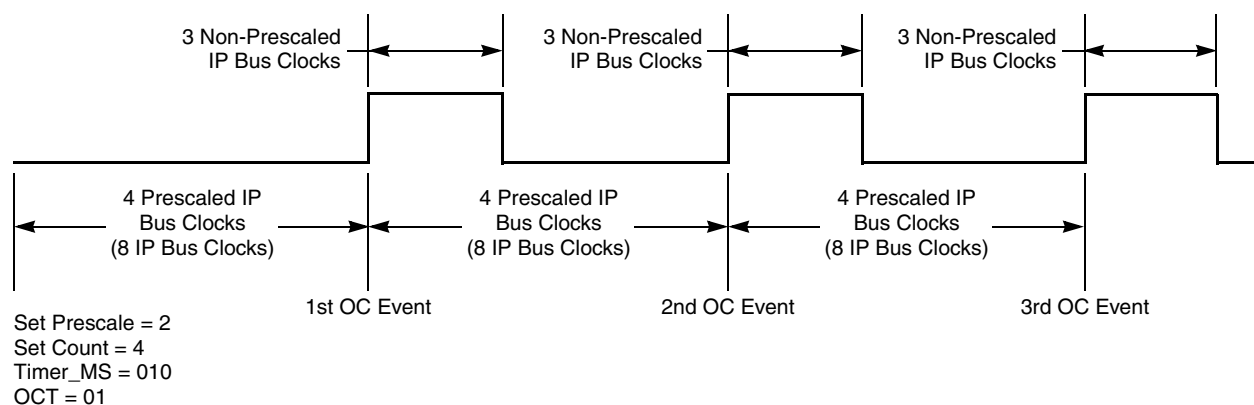


Figure 17-5. Output Pulse High Time Example

NOTE

It is the responsibility of the system software to set the Timer pin to the desired state before setting up the PRESCALE and COUNT fields in the GPT Counter Input and Updown Counter Output Register.

17.3.6 Output Pulse Low

A GPT Timer Channel can be programmed to issue a single pulse with negative polarity in response to an Output Compare Event. The OCT field and the TIMER_MS field of the GPT Enable and Mode Select Register must respectively be programmed to 10 and 010 to enable this mode. The LOW Time of the pulse, specified in non-prescaled IP Bus clocks, is programmed into the OCPW field of the GPT Enable and Mode Select Register. To create an Output Compare Event, a value can be written to the COUNT field of the GPT Counter Input and Updown Counter Output Register that specifies the time, expressed in prescaled IP Bus Clocks, when the next Output Compare event occurs. For example, if the OCPW field is written to 3, the PRESCALE field is written to 2 and the Count field is written to four the result is a positive pulse which is 3 IP Bus clocks wide that occurs eight (Prescale = 2, Count = 4) IP Bus clocks after writing the TIMER_MS field of the GPT Enable and Mode Select Register to 010. If the STOP_CONT bit is set to 1, continuous operation, the following waveform is generated.

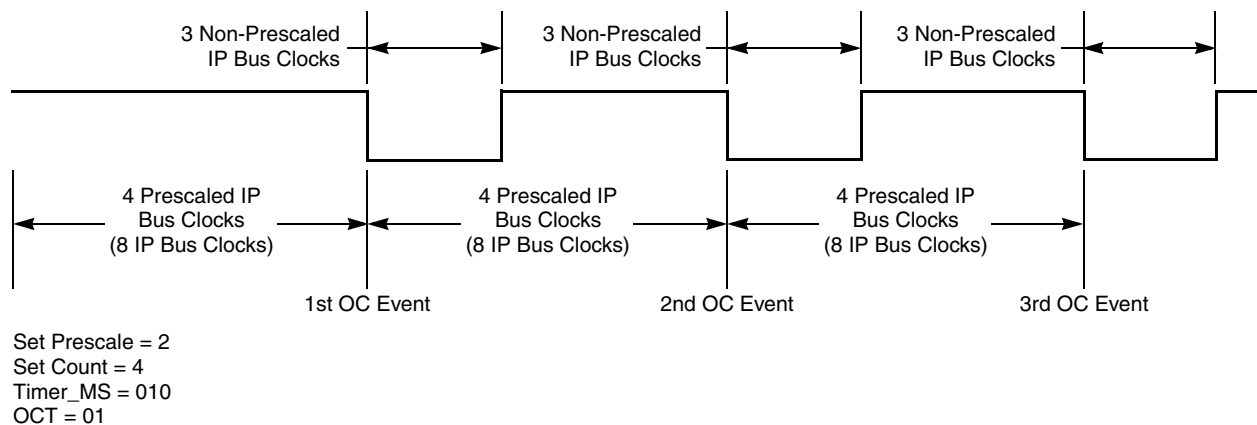


Figure 17-6. Output Pulse Low Time Example

NOTE

It is the responsibility of the system software to set the Timer pin to the desired state before setting up the PRESCALE and COUNT fields in the GPT Counter Input and Updown Counter Output Register.

17.3.7 Output Toggle

A GPT Timer Channel can be programmed to toggle from its present value in response to an Output Compare Event. The OCT field and the TIMER_MS field of the GPT Enable and Mode Select Register must respectively be programmed to 11 and 010 to enable this mode.

To create an Output Compare Event, a value can be written to the COUNT field of the GPT Counter Input and Updown Counter Output Register that specifies the time, expressed in prescaled IP Bus Clocks, when the next Output Compare event occurs. If the PRESCALE field is written to 5 and the Count field is written to 6 the result is a transition at 30 (Prescale = 5, Count = 6) IP Bus clock intervals after setting the OCT field to 11 and the Timer_MS field of the GPT Enable and Mode Select Register to 010. If the STOP_CONT bit is set to 1, continuous operation, the following wave form is generated. If the STOP_CONT bit is set to 0, only one Output compare Event occurs and the Timer Channel pin toggles only once.

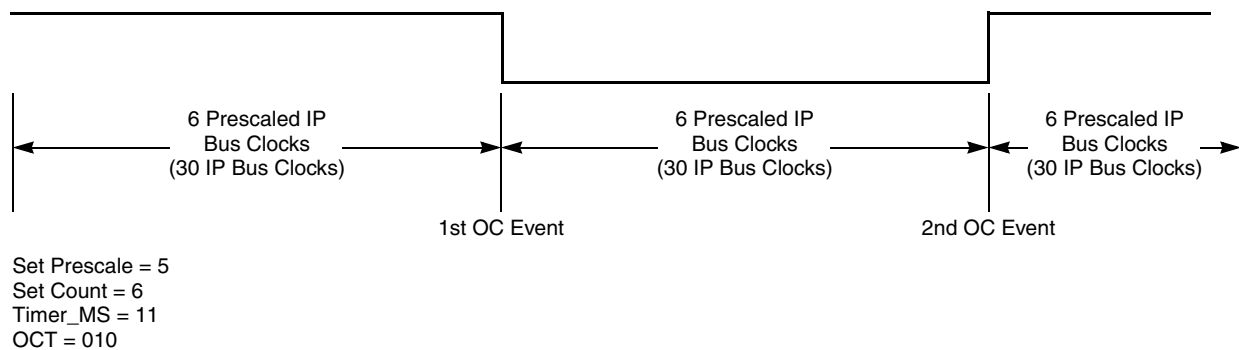


Figure 17-7. Output Compare Toggle Example

NOTE

It is the responsibility of the system software to set the Timer pin to the desired state before setting up the PRESCALE and COUNT fields in the GPT Counter Input and Updown Counter Output Register.

17.3.8 Pulse Width Modulation

In this mode the I/O pin is an Output. The user can program Period and Width values to create an adjustable, repeating output waveform on the I/O pin. A CPU interrupt can be generated at the beginning of each PWM Period, at which time a new Width value can be loaded. The new Width value, which represents ON time, is automatically applied at the beginning of the next period. There is no interrupt at the beginning of the first PWM Period. This mode is suitable for PWM audio encoding.

The ON TIME for the PWM signal is programmed into the WIDTH field of the GPT PWM Configuration Register in prescaled IP Bus Clocks. The Period of the PWM signal is programmed into the COUNT field of the GPT Counter Input Register. The PRESCALE field of the GPT Counter Input Register applies to both the COUNT value and the WIDTH Value. The ON TIME overlays the total period. That is, the WIDTH field determines the total period of the PWM signal.

In the following example, the Prescale field equals 8, the Count field equals 6 and the Width field equals 2.

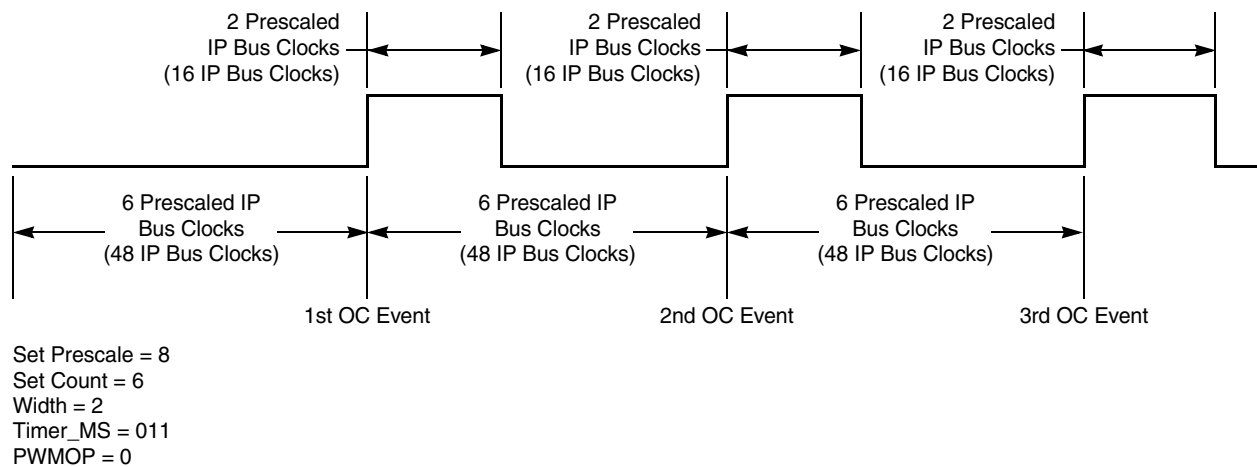


Figure 17-8. PWM Example with OFF TIME = LOW

NOTE

It is the responsibility of the system software to set the logic level on the TIMER pin to the desired value before involving the PWM Mode.

NOTE

When the TIMER_MS field is set to 011 to start the PWM signal, the first period is at the logic level specified by the PWMOP bit. The active pulse whose width is specified by the WIDTH field overlays the following period.

NOTE

In the present example, it is assumed that the TIMER pin is at a logic 0 at the time that the TIMER_MS field is written. From a practical standpoint, this effectively means that the first active pulse occurs one full PWM period after writing the TIMER_MS field. If the TIMER pin is at a logic 1 at the time that the TIMER_MS field is written, it appears as though the first pulse is one full PWM period wide. Therefore, it is important to program the TIMER pin such that it is in a known state before writing to the TIMER_MS field.

The following example is similar to the example shown in [Figure 17-8](#) with the exception that the OFF TIME is HIGH. In this case, PRESCALE = 9, COUNT = 4, WIDTH = 3, and PWMOP = 1.

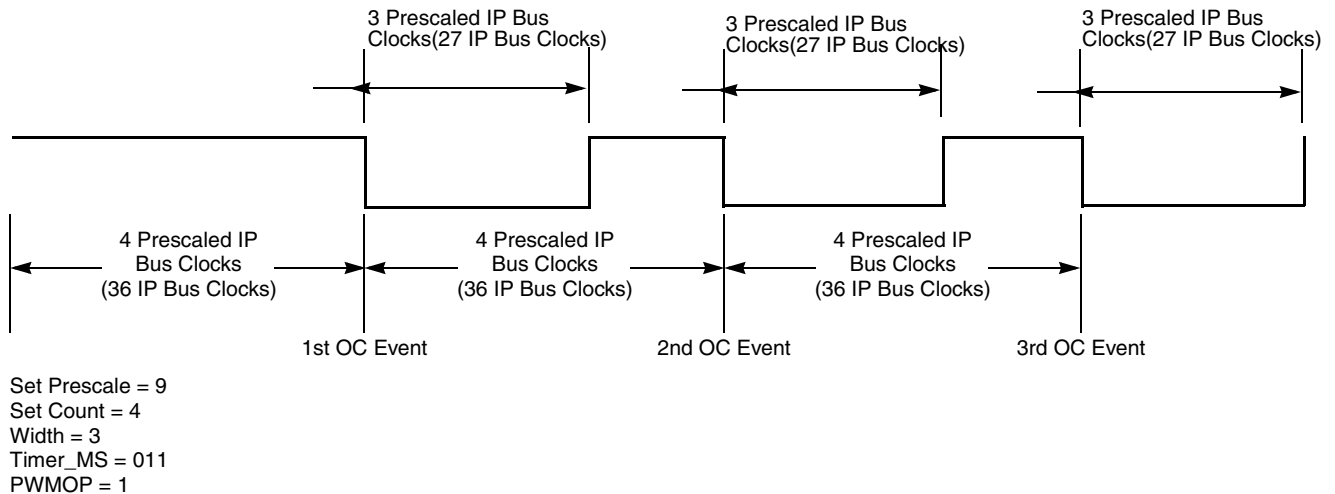


Figure 17-9. PWM Example with OFF TIME = HIGH

NOTE

It is the responsibility of the system software to set the logic level on the TIMER pin to the desired value before involving the PWM Mode.

NOTE

When the TIMER_MS field is set to 011 to start the PWM signal, the first period is at the logic level specified by the PWMOP bit. The active pulse whose width is specified by the WIDTH field overlays the following period.

NOTE

In the present example, it is assumed that the TIMER pin is at a logic 1 at the time that the TIMER_MS field is written. From a practical standpoint, this effectively means that the first active pulse occurs one full PWM period after writing the TIMER_MS field. If the TIMER pin is at a logic 0 at the time that the TIMER_MS field is written, it appears as though the first pulse is one full PWM period wide. Therefore, it is important to program the TIMER pin such that it is in a known state before writing to the TIMER_MS field.

17.3.9 Simple GPIO

In this mode the I/O pin operates as a GPIO pin. Each Timer pin can individually be specified as Input or Output, according to the programmable GPIO field. GPIO mode is mutually exclusive of Input Capture, Output Compare and PWM modes. That is, in the GPIO mode, the TIMER pin cannot be used for input capture or to output a timer waveform. In GPIO mode, CPU Timer modes remain available.

17.3.9.1 CPU Timer

The I/O pin is not used in this mode. After enabled, the counters run until they reach a programmed Terminal Count. When this occurs, an interrupt can be generated to the CPU. This Timer mode can be used simultaneously with the Simple GPIO mode.

Chapter 18

General Purpose I/O (GPIO)

18.1 Introduction

This chapter describes the general purpose I/O module, including pin descriptions, register settings and interrupt capabilities.

The GPIO module supports 28 general-purpose I/O pins. Each pin can be configured as an input or as an output. The module also supports 4 general purpose input pins. If a pin is configured as an input, it can optionally generate an interrupt upon detection of a change in state. If a pin is configured as an output, it can be configured as an open-drain output or a fully active output. When a GPIO pin is configured as an input, it can serve as a DMA request signal.

See [Figure 18-1](#).

18.2 Features

The GPIO unit implements the following features:

- Thirty-two input/output pins
- All pins are configured as inputs when the MPC5121e reset signal is asserted
- Open-drain capability on all pin
- All pins, when configured as inputs, can optionally generate an interrupt upon detection of a change of state.
- When a GPIO pin is configured as an output, its DMA request functionality is disabled

The following sections provide an overview and detailed descriptions of the GPIO signals.

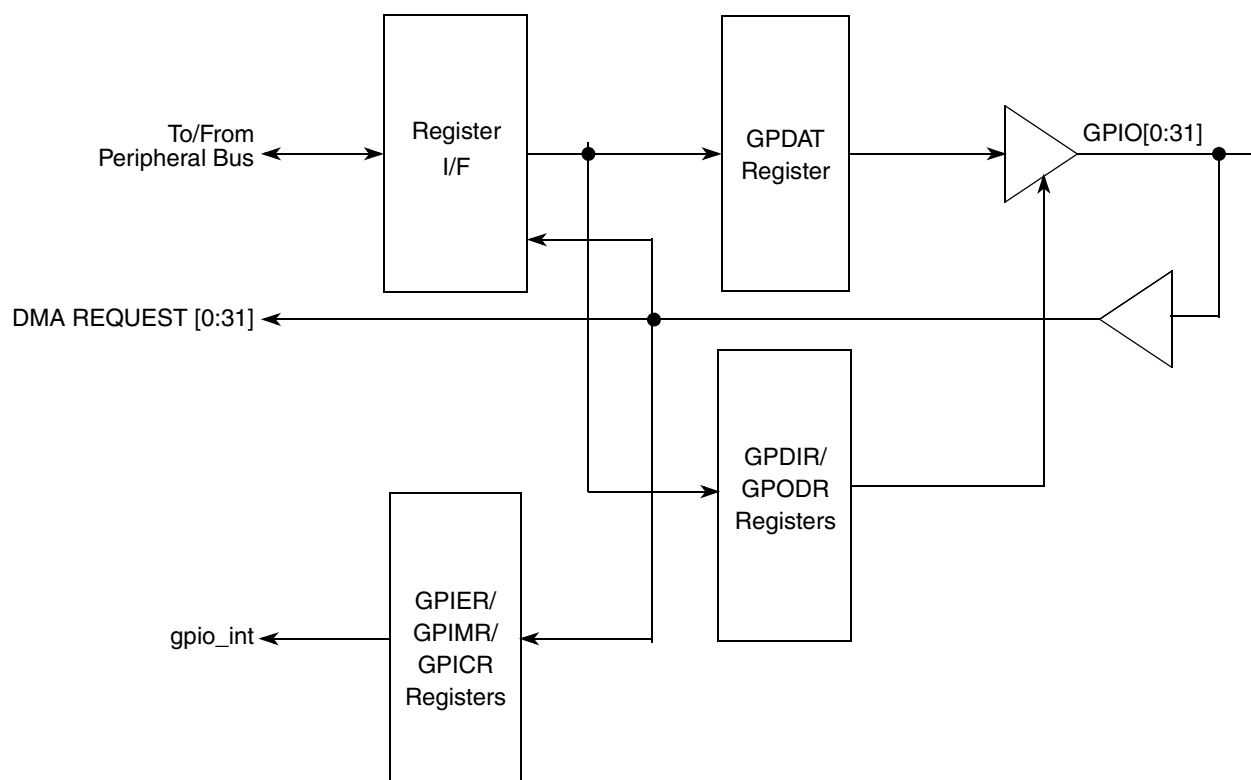


Figure 18-1. GPIO Module Block Diagram

18.3 Memory Map/Register Definition

All GPIO registers are 32 bits wide and are located on 32-bit address boundaries. All addresses used in this chapter are offsets from the GPIO module base address. A GPIO module memory map is shown in [Table 18-1](#). Reading undefined portions of the memory map returns all zeros; writing has no effect.

Table 18-1. GPIO Register Address Map

Offset	Register	Access	Section/ Page
0x00	GPIO Direction Register (GPDIR)	R/W	18.3.1.1/18-3
0x04	GPIO Open Drain Register (GPODR)	R/W	18.3.1.2/18-4
0x08	GPIO Data register (GPDAT)	R/W	18.3.1.3/18-5
0x0C	GPIO Interrupt Event Register (GPIER)	R/W	18.3.1.4/18-6
0x10	GPIO Interrupt Mask Register (GPIMR)	R/W	18.3.1.5/18-7
0x14	GPIO External Interrupt Control Register 1 (GPICR1)	R/W	18.3.1.6/18-7
0x18	GPIO External Interrupt Control Register 2 (GPICR2)	R/W	18.3.1.6/18-7

18.3.1 Register Descriptions

18.3.1.1 GPIO Direction Register (GPDIR)

The GPIO direction register (GPDIR) shown in [Figure 18-2](#) defines the direction of individual GPIO pins.

NOTE

Bits D0–D31 set the I/O state of the GPIO 0–31 pins.

Register address: 0x00

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x02

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-2. GPIO Direction Register (GPDIR)

Table 18-2. GPDIR Field Descriptions

Field	Description
D[0:31]	Direction. Indicates whether a pin is used as an input or an output. 0 The corresponding pin is an input. 1 The corresponding pin is an output.

Figure 18-3.

18.3.1.2 GPIO Open Drain Register (GPODR)

The GPIO open drain register (GPODR) shown in Figure 18-4 defines the individual GPIO pins output drive structure.

NOTE

Bits D0 – D31 set the drive structure of the GPIO 0 – 31 pins.

Register address: 0x04

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x06

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-4. GPIO Open Drain Register (GPODR)

Table 18-3. GPODR Field Descriptions

Field	Description
D[0:31]	Output drive configuration. Indicates whether a pin is actively driven as an output or is an open-drain driver. 0 The I/O pin is actively driven as an output. 1 The I/O pin is an open-drain driver. As an output, the pin is driven active-low, otherwise it is tri-stated.

Figure 18-5.

18.3.1.3 GPIO Data Register (GPDAT)

The GPIO data register (GPDAT) shown in [Figure 18-6](#) carries the data in/out for individual GPIO pins.

NOTE

Bits D0 – 31 are driven on the GPIO 0 – 31 pins when configured as outputs.

Bits D0 – 31 reflect the state of GPIO 0 – 31 pins when configured as inputs.

Register address: 0x08

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x0A

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-6. GPIO Data Register (GPDAT)

Table 18-4. GPDAT Field Descriptions

Field	Description
D[0:31]	Data. Write data is latched and presented on external pins if GPDIR has configured the GPIO pin as an output. Read operation always returns the data at the pin.

Figure 18-7.

18.3.1.4 GPIO Interrupt Event Register (GPIER)

The GPIO interrupt event register (GPIER) shown in Figure 18-8 carries information about the events that cause an interrupt. Each bit in the interrupt event register (GPIER), corresponds to an individual interrupt source. GPIER bits are cleared by writing ones. Writing zero has no effect.

NOTE

Bits D0 – D31 are set in response to interrupt events occurring on GPIO 0 – 31 pins.

Register address: 0x0C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
W																
Reset ¹	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x0E

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
W																
Reset ¹	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This register should be initialized by writing the following: 0x FF FF FFFF.

Figure 18-8. GPIO Interrupt Event Register (GPIER)

Table 18-5. GPIER Field Descriptions

Field	Description
D[0:31]	Interrupt events. Indicates whether the interrupt event occurred on the corresponding GPIO pin. 0 No interrupt event occurred on the corresponding GPIO pin. 1 Interrupt event occurred on the corresponding GPIO pin.

Figure 18-9.

18.3.1.5 GPIO Interrupt Mask Register (GPIMR)

The GPIO interrupt mask register (GPIMR) shown in Figure 18-10 defines the interrupt masking for the individual GPIO pins. When a masked interrupt occurs, the corresponding GPIER bit is set, regardless of the GPIMR state. When one or more non-masked interrupt events occur, the GPIO module issues an interrupt to the Power Architecture core.

NOTE

Bits D0 – D31 mask interrupts from GPIO 0 – 31 pins.

Register address: 0x10

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x12

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-10. GPIO Interrupt Mask Register (GPIMR)

Table 18-6. GPIMR Field Descriptions

Field	Description
D[0:31]	Interrupt mask. Indicates whether an interrupt event is masked or non-masked. 0 The input interrupt pin is masked (disabled). 1 The input interrupt pin is non-masked (enabled).

Figure 18-11.

18.3.1.6 GPIO Interrupt Control Register 1 and 2 (GPICR1 and GPICR2)

The GPIO interrupt control register 1 (GPICR1) and GPIO interrupt control register 2 (GPICR2) shown in Figure 18-13 determines which type of event causes each individual GPIO pin to set their associated bit in the GPIO interrupt event register and, if enabled, causes an interrupt to be asserted to the CPU.

The D [0:31] fields of the GPIO interrupt control registers specify which type of event causes an interrupt for GPIO [0:31]. The interrupt function for each GPIO pin is individually programmable. For example, GPIO0 is controlled by the D0 field, GPICR1[0:1]; GPIO16 is controlled by the D16 field, GPICR2[0:1].

NOTE

Bit fields D0 – D31 specify the interrupt event type for GPIO 0 – 31 pins.

Register address: 0x14

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D0				D1				D2				D3			
W	D0				D1				D2				D3			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x16

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D8				D9				D10				D11			
W	D8				D9				D10				D11			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-12. GPIO Interrupt Control Register (GPICR1)

Register address: 0x18

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D16				D17				D18				D19			
W	D16				D17				D18				D19			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register address: 0x1a

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	D24				D25				D26				D27			
W	D24				D25				D26				D27			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-13. GPIO Interrupt Control Register (GPICR2)

Table 18-7. GPICR1 and GPICR2 Field Descriptions

Field	Description
D[0:31]	Edge detection mode. The corresponding GPIO pin asserts an interrupt request according to the following: 00 Any change on the state of the GPIO pin generates an interrupt request. 01 Low-to-high change on the GPIO pin generates an interrupt request. 10 High-to-low change on the GPIO pin generates an interrupt request. 11 Pulse (any 2 transitions) on the GPIO pin generates an interrupt request.

18.4 Functional Description

The GPIO module supports 32 general purpose I/O pins. Each GPIO pin can be configured as an input or output. If a GPIO pin is configured as an input, the pin can optionally generate an interrupt upon the detection of a change in state. If the GPIO interrupt control register bits for a particular GPIO pin are configured as 00, the GPIO pin detects any change of state, set its corresponding bit in the GPIO interrupt event register (GPIER) and, if enabled, generate an interrupt. If the GPIO interrupt control register bits for a particular GPIO pin are configured as 01 or 10, the GPIO pin respectively detects a low-to-high transition or a high-to-low transition, set the corresponding bit in the GPIO interrupt event register (GPIER) and, if enabled, generate an interrupt. As an input, GPIO 0 through 30 can serve as a DMA requestor. GPIO31 cannot act as a DMA requestor. If a GPIO pin is configured as an output, it can be individually configured as an open-drain or a fully active output.

Chapter 19

IIM/Fusebox

19.1 Introduction

The IC Identification Module (IIM) provides an interface for reading and programming information stored in on-chip fuse elements.

19.2 Overview

The IIM provides the primary user-visible mechanism for interfacing with on-chip fuse elements. Among other uses, e-fuses can be used for unique chip identifiers, cryptographic keys, and various control signals requiring permanent non-volatility.

The IIM consists of a master controller, a fuse value shadow cache, and a set of registers to hold the values of signals visible outside the module. Two 256 bits fuse banks are implemented on the MPC5121e.

The e-Fuses may be blown under software control at the customer factory or in the field. They include a mechanism to inhibit further blowing of fuses (write-protect) to support secure computing environments. The fuse values may also be overridden by software without modifying the fuse element. Similar to the write-protect functionality, the override functionality can also be permanently disabled.

19.2.1 Features

- Two fuse banks, each fuse bank size is 256 bits
- Ability to write-protect e-Fuses on a per-bank basis

19.2.2 Modes of Operation

The IIM is in its functional mode (all specified functionality available) any time it is out of reset and supplied with the proper clocks.

For programming the external FUSE programming supply, AVDD_FUSEWR must be applied.

19.3 Memory Map and Register Definition

[Section 19.3.2, “Register Descriptions”](#) provides the detailed register descriptions for all of the IIM registers

19.3.1 Memory Map

All registers are 8-bit wide, but addressable on 32-bit boundaries. Only the bottom 8 bits (the usable bits) of each register are shown in the following diagrams. The top 24 bits always read as 0 and writes to them are ignored. [Table 19-1](#) shows the IIM memory map.

Table 19-1. IIM Memory Map

Address Offset	Register	Access	Section/Page
0x000	STAT – Status register	R/W	
0x004	STATM – Status IRQ Mask register	R/W	
0x008	ERR – Module Errors register	R/W	
0x00C	EMASK – Error IRQ Mask register	R/W	
0x010	FCTL – Fuse Control register	R/W	
0x014	UA – Upper Address register	R/W	
0x018	LA – Lower Address register	R/W	
0x01C	SDAT – Explicit Sense Data register	Read-only	
0x028	PREG_P – Program Protection register	R/W	
0x03C	DIVIDE – Divide Factor register	R/W	
0x0C00	FBAC1 – Fuse bank 1 Protection Register	R/W	
0x0C04 ... 0x0C7C	FB1W1 – Fuse bank 1 Data (available for user)	R/W	

19.3.2 Register Descriptions

This section contains the detailed register descriptions for the IIM registers.

19.3.2.1 Status Register (STAT)

See [Figure 19-1](#) for illustration of valid bits in the status register and [Table 19-2](#) for description of the bit fields.

Offset IIM_BASE +0x00Access: Supervisor read-only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									BUSY						PRGD	SNSD
W															w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	—	—	—	—	—	0

= Unimplemented or Reserved

Figure 19-1. Status Register (STAT)

Table 19-2. STAT Field Descriptions

Field	Description
BUSY	Indicates whether the IIM is busy with a program or sense cycle. Any attempt to access the IIM registers other than STAT while it is busy with a program or sense cycle (BUSY asserted) results in a bus error. 0 The IIM is not busy with a program or sense cycle 1 The IIM is busy with a program or sense cycle
PRGD	Program Done. Indicates an e-Fuse program operation is done. Assertion causes an interrupt request if PRGD_M is set in the status IRQ mask register. This bit is automatically set by hardware upon completion of an e-Fuse program cycle; software must clear the bit by writing 1 to it. 0 Program operation has not finished (read); no meaning (write) 1 Program operation has finished (read); clear bit (write)
SNSD	Explicit Sense Cycle Done. Indicates that an explicit fuse sense cycle is done, and the data is available in SDAT. Assertion causes an interrupt request if SNSD_M is set in the status IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it. 0 No explicit sense cycle has finished (read); no meaning (write) 1 An explicit sense cycle has finished (read); clear bit (write)

19.3.2.3 Module Errors Register (ERR)

See [Figure 19-3](#) for illustration of valid bits in the module errors register and [Table 19-4](#) for description of the bit fields.

Offset IIM_BASE +0x08 Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W										WPE	OPE	RPE	WLRE	SNSE	PARITYE	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	—	u


 = Unimplemented or Reserved

Figure 19-3. Module Errors Register (ERR)

Table 19-4. NAME Field Descriptions

Field	Description
WPE	Write Protect Error. Indicates an e-Fuse program operation was attempted to a write-protected fuse bank, a locked words, or when the value of PRG_P is not 0XAA. Assertion causes an interrupt request if WPE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it. 0 There was no write-protect error (read); no meaning (write) 1 There was a write-protect error (read); clear bit (write)
OPE	Override Protect Error. Indicates an attempt was made to override the values in an override-protected fuse bank or a locked words. Assertion causes an interrupt request if OPE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it. 0 There was no override-protect error (read); no meaning (write) 1 There was an override-protect error (read); clear bit (write)
RPE	Read Protect Error. Indicates an attempt was made to read values from a read-protected fuse bank or SCC. Assertion causes an interrupt request if RPE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it. 0 There was no read-protect error (read); no meaning (write) 1 There was a read-protect error (read); clear bit (write)
WLRE	Write to Locked Register Error. Indicates an attempt was made to write to a locked SCS register. Assertion causes an interrupt request if WLRE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it. 0 There was no write-to-locked-register error (read); no meaning (write) 1 There was a write-to-locked-register error (read); clear bit (write)

Offset IIM_BASE +0x08 Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W										WPE	OPE	RPE	WLRE	SNSE	PARITYE	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	—	u

= Unimplemented or Reserved

Figure 19-3. Module Errors Register (ERR)

Table 19-4. NAME Field Descriptions (continued)

Field	Description
SNSE	<p>Explicit Sense Cycle Error. Indicates that an explicit fuse sense was refused, because FBESP is set to 1 or more than two bits of SNS_N, SNS_1, SNS_0, and PRG are asserted at the same moment. Assertion causes an interrupt request if SNSE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it.</p> <p>0 There was no explicit sense error (read); no meaning (write)</p> <p>1 There was an explicit sense error (read); clear bit (write)</p>
PARITYE	<p>Parity Error of Cache. Indicates that a parity error was detected in the hardware or software fuse cache. Assertion causes an interrupt request if PARITYE_M is set in the errors IRQ mask register. This bit is automatically set by hardware and must be cleared by software by writing 1 to it.</p> <p>0 There was no parity error (read); no meaning (write)</p> <p>1 There was an parity error (read); clear bit (write)</p>

19.3.2.4 Error IRQ Mask Register (EMASK)

See [Figure 19-4](#) for illustration of valid bits in the error IRQ mask register and [Table 19-5](#) for description of the bit fields.

Offset IIM_BASE +0x0CAccess: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R										WPE_	OPE_	RPE_	WLRE_	SNSE_	PARITYE	
W										M	M	M	M	M	_M	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 19-4. Error IRQ Mask Register (EMASK)

Table 19-5. EMASK Field Descriptions

Field	Description
WPE_M	Write Protect Error Mask. Masks or unmask IRQ generation due to WPE events. 0 WPE events do not cause an IRQ 1 WPE events cause an IRQ
OPE_M	Override Protect Error Mask. Masks or unmask IRQ generation due to OPE events. 0 OPE events do not cause an IRQ 1 OPE events cause an IRQ
RPE_M	Read Protect Error Mask. Masks or unmask IRQ generation due to RPE events. 0 RPE events do not cause an IRQ 1 RPE events cause an IRQ
WLRE_M	Write to Locked Register Error Mask. Masks or unmask IRQ generation due to WLRE events. 0 WLRE events do not cause an IRQ 1 WLRE events cause an IRQ
SNSE_M	Explicit Sense Cycle Error Mask. Masks or unmask IRQ generation due to SNSE events. 0 SNSE events do not cause an IRQ 1 SNSE events cause an IRQ
PARITYE_M	Parity Error of Cache Mask. Masks or unmask IRQ generation due to PARITYE events. 0 PARITYE events do not cause an IRQ 1 PARITYE events cause an IRQ

19.3.2.5 Fuse Control Register (FCTL)

See [Figure 19-5](#) for illustration of valid bits in the fuse control register and [Table 19-6](#) for description of the bit fields.

Offset IIM_BASE +0x10Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R										PRG_LENGTH[2:0]			ESNS_N	ESNS_0	ESNS_1	PRG
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 19-5. Fuse Control Register (FCTL)

Table 19-6. FCTL Field Descriptions

Field	Description
PRG_LENGTH [2:0]	Program Length. These three bits define the length of program pulse. PRG_LENGTH*(period of 32k clock)
ESNS_N	Explicit Sense—Normal. Writing 1 to this bit initiates an unstressed (normal) explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. Only one of ESNS_N, ESNS_0, ESNS_1, and PRG can be asserted. Otherwise, ESNS_E is asserted to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate an unstressed explicit sense cycle (write)
ESNS_0	Explicit Sense—0 Stressed. Writing 1 to this bit initiates a 0-stressed explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. During 0-stressed explicit sense cycles, the EPM_READSENSE0 signal is asserted to the fuse banks. Only one of ESNS_N, ESNS_0, ESNS_1, and PRG can be asserted. Otherwise, ESNS_E is asserted to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a 0-stressed explicit sense cycle (write)
ESNS_1	Explicit Sense—1 Stressed. Writing 1 to this bit initiates a 1-stressed explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. During 1-stressed explicit sense cycles, the EPM_READSENSE1 signal is asserted to the fuse banks. Only one of ESNS_N, ESNS_0, ESNS_1, and PRG can be asserted. Otherwise, ESNS_E is asserted to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a 1-stressed explicit sense cycle (write)
PRG	Program. Writing 1 to this bit initiates a fuse program cycle. Reading of this bit always returns zero. FSM generate a done signal when the operation complete. This bit is cleared automatically by hardware when program operation completes. Only one of ESNS_N, ESNS_0, ESNS_1, and PRG can be asserted. Otherwise, ESNS_E is asserted to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a program cycle (write)

19.3.2.6 Upper Address Register (UA)

This register contains the top part of the address of the e-Fuse bit to be programmed or the word to be sensed in an explicit sense cycle. Programming is done on a bit-basis, so the program address is a full-bit address. Sensing is done on a word (8-bit) basis, so the bottom three bits of the address are ignored.

See [Figure 19-6](#) for illustration of valid bits in the upper address register and [Table 19-7](#) for description of the bit fields.

Offset IIM_BASE +0x14Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													A[13:8]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 19-6. Upper Address Register (UA)

Table 19-7. UA Field Descriptions

Field	Description
A[13:8]	The top six bits of the address of the e-Fuse bit to be programmed or the word to be sensed explicitly. The address must be written prior to setting the PRG or ESNS_x bit in FCTL to initiate the program/sense operation. A[13:11] select the fuse bank. A[10:8] provide the most significant portion of the row address within the bank.

19.3.2.7 Lower Address Register (LA)

This register contains the bottom eight bits of the address of the e-Fuse bit to be programmed or word to be explicitly sensed.

See [Figure 19-7](#) for illustration of valid bits in the lower address register and [Table 19-8](#) for description of the bit fields.

Offset IIM_BASE +0x18Access: Supervisor read/write

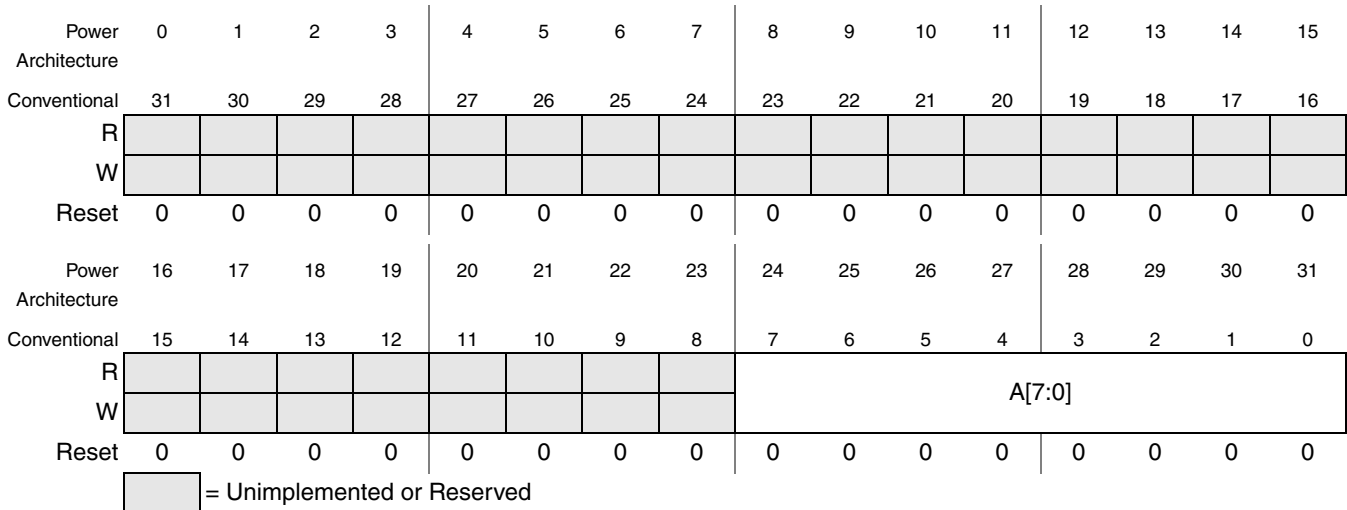


Figure 19-7. Lower Address Register (LA)

Table 19-8. UA Field Descriptions

Field	Description
A[7:0]	The bottom eight bits of the address of the e-Fuse bit to be programmed or word to be sensed explicitly. The address must be written prior to setting the PRG or ESNS_x bit in FCTL to initiate a program or sense operation. A[7:3] provides the least significant portion of the row address. A[2:0] select the bit position within the selected row.

19.3.2.8 Explicit Sense Data Register (SDAT)

See [Figure 19-8](#) for illustration of valid bits in the explicit sense data register and [Table 19-9](#) for description of the bit fields.

Offset IIM_BASE +0x1CAccess: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									D[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 19-8. Explicit Sense Data Register (SDAT)

Table 19-9. SDAT Field Descriptions

Field	Description
D[7:0]	The data sensed from the fuses.

On an explicit sense cycle, the data sensed from the fuses is placed in this register at the conclusion of the sense cycle. Software can recognize the conclusion of the sense cycle by the assertion of SNSD in STAT.

19.3.2.9 Program Protection Register (PRG_P)

This register is to prevent accidental fuse programming. The fuses can be blown only when the value of this register is 0xAA. Software should only program this register to 0xAA while actively blowing fuses. After the program operation is complete, this register should be immediately reprogrammed to a different value.

See [Figure 19-9](#) for illustration of valid bits in the program protection register and [Table 19-10](#) for description of the bit fields.

Offset IIM_BASE +0x28 Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									PROTECTION_REG[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 19-9. Program Protection Register (PRG_P)

Table 19-10. PRG_P Field Descriptions

Field	Description
PROTECTION_REG[7:0]	The fuses can be blown only when the value of this register is 0xAA. Any attempt to program the fuse while the value is other than 0xAA is terminated with error, and the WPE bit is asserted.

19.3.2.10 Divide Factor Register (DIVIDE)

This register contains the divide factor to generate a 32 kHz clock for programming the fuses from the IPS clock. The unit of IPS clock is MHz.

See [Figure 19-10](#) for illustration of valid bits in the divide factor register and [Table 19-11](#) for description of the bit fields.

Offset IIM_BASE +0x3CAccess: Supervisor read/write

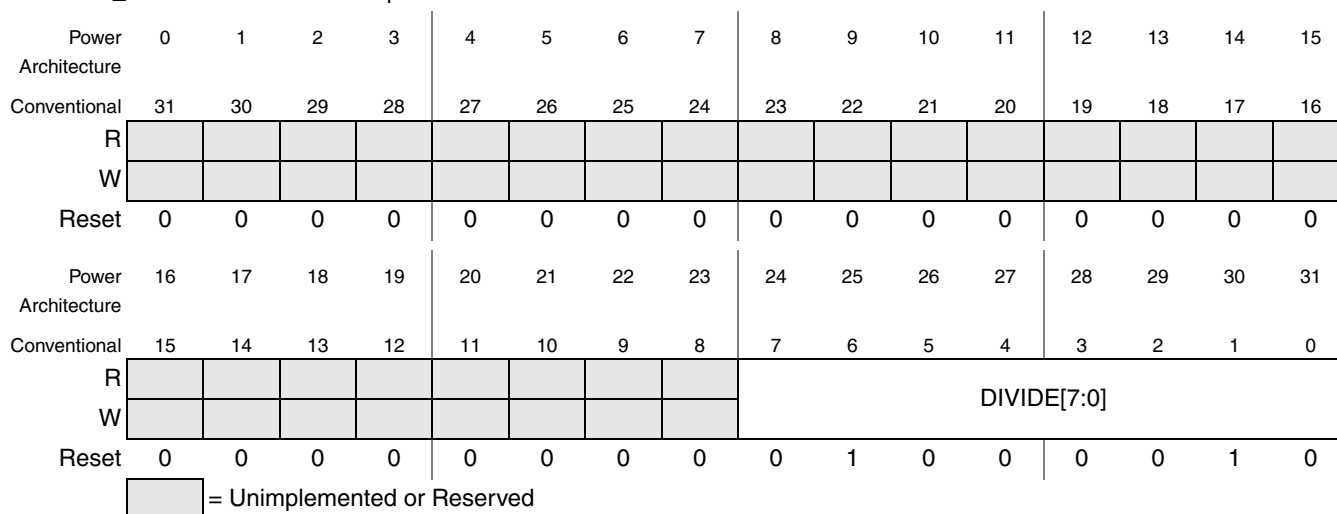


Figure 19-10. Divide Factor Register (DIVIDE)

Table 19-11. DIVIDE Field Descriptions

Field	Description
DIVIDE[7:0]	This register provides a divide factor with which IPG_CLK can be divided to an 1 MHz clock. Then the 1 MHz clock is divided to 32 kHz. Its value should be the frequency of IPS clock (unit: MHz), but the minimum is 1. The default value is 0x42.

19.3.2.11 Fuse Bank 1 Protection Register (FBAC1)

This register corresponds to the first word in fuse bank 1, which holds the fuse bank access protection information. The fuses associated with this register must be sensed out when IIM come out of reset.

See [Figure 19-11](#) for illustration of valid bits in the fuse bank 1 access protection register and [Table 19-12](#) for description of the bit fields.

Offset IIM_BASE +0x0C00Access: Supervisor read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									FBWP	FBOP	FBRP	FBSP	FBESP			
W																
Reset	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

□ = Unimplemented or Reserved

Figure 19-11. Fuse Bank 1 Protection Register (FBAC1)

Table 19-12. FBAC1 Field Descriptions

Field	Description
FBWP	Fuse Bank Write Protect. Controls whether this fuse bank (Fuse Bank 1) may be programmed. 0 (Unblown) = Fuse bank 1 may be programmed 1 (Blown) = Fuse bank 1 may not be programmed (it is write-protected)
FBOP	Fuse Bank Override Protect. Controls whether this fuse bank (Fuse Bank 1) may be overridden. 0 (Unblown) = Fuse bank 1 may be overridden 1 (Blown) = Fuse bank 1 may not be overridden (it is override-protected)
FBRP	Fuse Bank Read Protect. Controls whether this fuse bank (Fuse Bank 1) may be read. 0 (Unblown) = Fuse bank 1 may be read by software 1 (Blown) = Fuse bank 1 may not be read by software (it is read-protected)
FBSP	Reserved
FBESP	Fuse Banks Explicit Sense Protect. Controls whether this fuse bank (Fuse Bank 1) may be explicitly sensed. The state of this fuse controls whether the IIM state machine allow explicit sense cycles (normal, 0-stress, or 1-stress). 0 (Unblown) = Fuse bank 1 may be explicitly sensed by software 1 (Blown) = Fuse bank 1 may not be explicitly sensed by software (it is sense-protected)

Note: Reading these bits returns the fuse state (0 = unblown; 1 = blown) so long as FBAC1[FBRP] is 0 (unblown). Disallowed reads always return 0 and cause ERR[RPE] to be set. Writing these bits overrides the values without modifying the fuse elements. Overriding is allowed so long as FBAC1[FBOP] is 0 (unblown). Disallowed attempts to override are ignored and cause ERR[OPE] to be set. The corresponding fuse elements may be programmed (blown) using the fuse programming sequence, so long as FBAC1[FBWP] is 0 (unblown). Disallowed attempts to program fuses are ignored and cause ERR[WPE] to be set.

19.3.2.12 Fuse Bank 1 Data Register (FB1W1)

Offset IIM_BASE +0x0C04 – 0x0C7C Access: Supervisor read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									DATA[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—


 = Unimplemented or Reserved

Figure 19-12. Fuse Bank 1 Data Register (FB1W1)

Table 19-13. FB1W1 Field Descriptions

Field	Description
DATA[7:0]	Reading these bits returns the fuse state (0 = unblown; 1 = blown) so long as FBAC1[FBRP] is 0 (unblown). Disallowed reads always return 0 and cause ERR[RPE] to be set. Writing these bits overrides the values without modifying the fuse elements. Overriding is allowed so long as FBAC1[FBOP] equals zero. Disallowed attempts to override are ignored and cause ERR[OPE] to be set. The corresponding fuse elements may be programmed (blown) using the fuse programming sequence, so long as FBAC1[FBWP] is 0. Disallowed attempts to program fuses are ignored and cause ERR[WPE] to be set.

19.4 Functional Description

The IIM consists of a master controller, a fuse value shadow cache, and a set of registers to hold the values of signals visible outside the module. Two 256 bits fuse banks are implemented on the MPC5121e.

Program operations are done on a bit basis. For programming, the external FUSE programming supply AVDD_FUSEWR must be applied.

19.4.1 Fuse Bank 0

Fuse bank 0 (256 bits) is reserved for Freescale internal use only.

19.4.2 Fuse Bank 1

Fuse bank 1 (256 bits) is available for user data.

Chapter 20

Integrated Programmable Interrupt Controller (IPIC)

20.1 Introduction

This chapter describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with some programming guidelines. The MPC5121e IPIC unit prioritizes and manages interrupts from the following controller units:

- DDR memory controller (DDR)
- LocalPlus controller (LPC)
- PCI
- Four-channel PCI DMA controller (DMA)
- Message unit (MU)
- Ethernet controller (FEC)
- Programmable serial controllers (PSC)
- USB controllers (USB)
- NAND flash controller (NFC)
- Parallel ATA controller (PATA)
- System bus arbiter (SBA)
- Real time clock timer (RTC)
- Eight global timers (GTM)
- Software watchdog timer (WDT)
- I²C controllers (I²C)
- Controller area network (CAN)
- Byte data link controller (BDLC)
- Audio accelerator (AXE)
- Display interface unit (DIU)
- Sony/Phillips digital interface (SPDIF)
- Secure digital host controller (SDHC)
- Direct memory access (DMA2)
- Power management controller (PMC)
- General-purpose IO controller (GPIO)
- Serial ATA controller (SATA)
- PCS FIFO controller (FIFOC)
- Graphics controller (MBX)¹

1. Not available in MPC5123

- Temperature sensor (TEMP)
- IC identification (IIM)
- Video In (VIU)

The interrupt sources controlled by the IPIC unit cause exceptions in the processor core. The internal interrupt signal (\overline{int}) is the main interrupt output from the IPIC to the Power Architecture core and causes the regular interrupt exception. The \overline{cint} signal is the critical interrupt output from the IPIC to the e300 core and causes the critical interrupt exception. The \overline{smi} signal is the system management interrupt output from the IPIC to the Power Architecture core and causes the system management interrupt exception. The machine check exception is caused by the internal \overline{mcp} signal generated by the IPIC, informing the host processor of error conditions, assertion of the external $\overline{IRQ0}$ machine-check request (enabled when SEMSR[SIRQ0]=1), and other conditions.

Figure 20-1 shows the relationship of the various functional blocks and external signals of the MPC5121e to the IPIC unit.

The IPIC receives interrupt request signals from the sources external and internal to the integrated device.

The unit selects the highest priority interrupt from all current interrupts and forwards it to the internal processor core.

The IPIC also manages an internal non-maskable machine-check processor signal (\overline{mcp}) and interrupt generated by the off-chip interrupt sources ($\overline{IRQ}[1:0]$).

The interrupt router of the IPIC monitors the outputs of the internal configuration registers. When the priority is highest in one of the received interrupt signals, the IPIC sets the corresponding bit in one of the interrupt pending registers (SIPNR or SEPNR). If the interrupt is not masked, the IPIC asserts the \overline{int} , \overline{cint} , or \overline{smi} signal to indicate an interrupt request to the processor. When the processor is executing the specific interrupt handler code, the processor must vectorize the external interrupt handler by explicitly reading (in software) the corresponding interrupt vector register (SIVCR, SCVCR or SMVCR). In response to this read, the IPIC unit returns the vector (associated with the interrupt source) to the interrupt handler routine. In addition, the handler can vectorize different branches of interrupt handling.

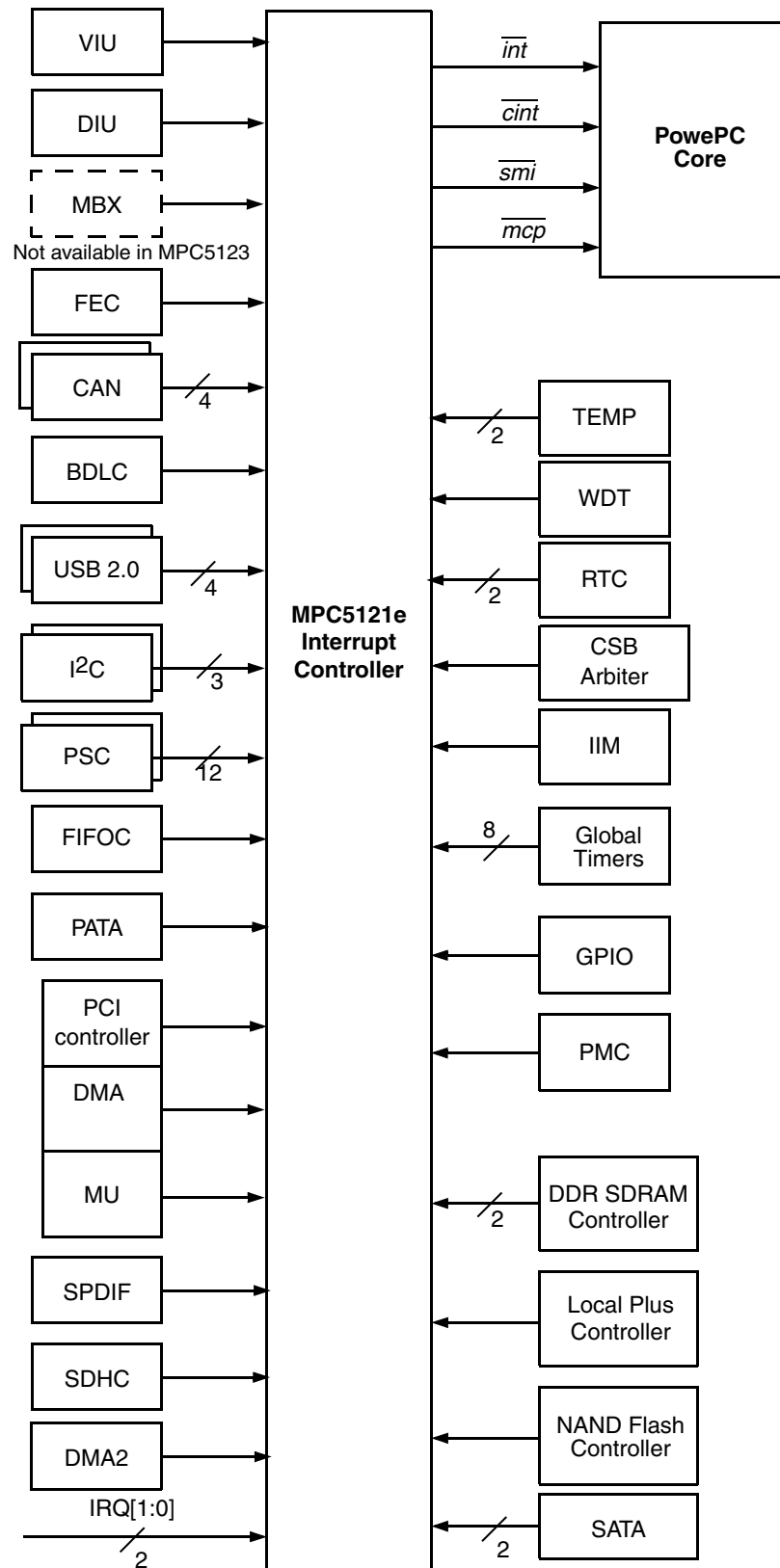


Figure 20-1. MPC5121e Interrupts Block Diagram

The IPIC can receive 65 separate interrupts from three different interrupt domains as follows:

- Two external interrupts – off-chip interrupt signals sources are $\overline{\text{IRQ}}[1:0]$
- Fifty-Eight internal interrupts – on-chip interrupt signals sources are DDR, LPC, NFC, PATA, PCI, DMA, MU, FEC, PSC, FIFO, USB, CSB arbiter, CAN, BDLC, DIU, AXE, SPDIF, SDHC, RTC, GTM, I²C, GPIO, GPT, SATA, MBX¹, TEMP, IIM, VIU, and PMC.
- 1 external and five internal non-maskable machine check exceptions – off-chip interrupt signal source can be $\overline{\text{IRQ0}}$. On-chip MCP interrupt signals sources are software watchdog timer (WDT), PCI, TEMP, and system bus arbiter (SBA).

The interrupt controller provides the ability to mask each interrupt source. Multiple events within GPIO or SBA peripheral event are also maskable.

When the IPIC receives an internal or external interrupt, its configuration register is checked to determine if it should be serviced as a normal external interrupt by the processor core (through the $\overline{\text{int}}$ signal). As a third alternative, if the incoming interrupt has been configured as a critical or system management interrupt, the IPIC completes the processing of the interrupt by asserting $\overline{\text{cint}}$ or $\overline{\text{smi}}$ to the core. The assertion of the $\overline{\text{cint}}$ or $\overline{\text{smi}}$ signal to the core causes the interrupt to be serviced as a critical or a system management interrupt, respectively.

20.1.1 Overview

The interrupt controller provides interrupt management responsible for receiving hardware-generated interrupts from different sources (both internal and external). It also prioritizes and delivers the interrupts to the CPU for servicing.

20.1.2 Features

The IPIC unit implements the following features:

- Supports two external and 58 internal discrete vectorized interrupt sources
- Supports one external and five internal machine check processor (MCP) interrupt sources
- Programmable highest priority request (can be programmed to support a critical ($\overline{\text{cint}}$) or system management ($\overline{\text{smi}}$) interrupt type)
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Four programmable priority internal groups of 8 on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Two highest priority interrupts from each group can be programmed to support a critical ($\overline{\text{cint}}$) or system management ($\overline{\text{smi}}$) interrupt type
- External and internal interrupts directed to host processor
- Unique vector number for each interrupt source

1. Not available in MPC5123

20.2 Memory Map/Register Definition

20.2.1 Register Summary

The IPIC programmable register map occupies 256 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All IPIC registers are 32 bits wide and they are located on 32-bit address boundaries. Software can perform byte, half-word, or word accesses to any IPIC registers. All addresses used in this chapter are offsets from the IPIC base, as defined in [Chapter 2, “System Configuration and Memory Map \(XLBMEN + Mem Map\)”](#).

[Table 20-1](#) shows memory map of the IPIC unit.

Table 20-1. IPIC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	System Global Interrupt Configuration Register (SICFR)	R/W	0x0000_0000	20.2.1.1/20-6
0x04	System Global Interrupt Vector Register (SIVCR)	R	0x0000_0000	20.2.1.2/20-8
0x08	System Internal Interrupt Pending Register (SIPNR_H)	R	0x0000_0000	20.2.1.3/20-11
0x0C	System Internal Interrupt Pending Register (SIPNR_L)	R	0x0000_0000	20.2.1.3/20-11
0x10	System Internal Interrupt Group A Priority Register (SIPRR_A)	R/W	0x0530_9770	20.2.1.4/20-13
0x14	System Internal Interrupt Group B Priority Register (SIPRR_B)	R/W	0x0530_9770	20.2.1.5/20-14
0x18	System Internal Interrupt Group C Priority Register (SIPRR_C)	R/W	0x0530_9770	20.2.1.6/20-15
0x1C	System Internal Interrupt Group D Priority Register (SIPRR_D)	R/W	0x0530_9770	20.2.1.7/20-16
0x20	System Internal Interrupt Mask Register (SIMSR_H)	R/W	0x0000_0000	20.2.1.8/20-17
0x24	System Internal Interrupt Mask Register (SIMSR_L)	R/W	0x0000_0000	20.2.1.8/20-17
0x28	System Internal Interrupt Control Register (SICNR)	R/W	0x0000_0000	20.2.1.9/20-19
0x2C	System External Interrupt Pending Register (SEPNR)	R/W	Special	20.2.1.10/20-21
0x30	System Mixed Interrupt Group A Priority Register (SMPRR_A)	R/W	0x0530_9770	20.2.1.11/20-22
0x34	System Mixed Interrupt Group B Priority Register (SMPRR_B)	R/W	0x0530_9770	20.2.1.12/20-23
0x38	System External Interrupt Mask Register (SEMSR)	R/W	0x0000_0000	20.2.1.13/20-24
0x3C	System External Interrupt Control Register (SECNR)	R/W	0x0000_0000	20.2.1.14/20-26
0x40	System Error Status Register (SERSR)	R/W	0x0000_0000	20.2.1.15/20-27
0x44	System Error Mask Register (SERMR)	R/W	0xE580_0000	20.2.1.16/20-28
0x48–0x4F	Reserved	—	—	
0x50	System Internal Interrupt Force Register (SIFCR_H)	R/W	0x0000_0000	20.2.1.17/20-29
0x54	System Internal Interrupt Force Register (SIFCR_L)	R/W	0x0000_0000	20.2.1.17/20-29
0x58	System External Interrupt Force Register (SEFCR)	R/W	0x0000_0000	20.2.1.18/20-31
0x5C	System Error Force Register (SERFR)	R/W	0x0000_0000	20.2.1.19/20-32
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	20.2.1.20/20-33

Table 20-1. IPIC Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	20.2.1.21/20-34
0x68– 0xFF	Reserved	—	—	

20.2.1.1 System Global Interrupt Configuration Register (SICFR)

SICFR, shown in [Figure 20-2](#), defines the highest priority interrupt and whether interrupts are grouped or spread in the priority table. See [Table 20-27](#) for more information.

Offset 0x00 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	HPI							0	MPSB	MPSA	0	IPSD	IPSC	IPSB	IPSA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	HPIT		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-2. System Global Interrupt Configuration Register (SICFR)
(The register is repeated for reference.)

Table 20-2. SICFR Field Descriptions

Field	Description
HPI	Highest Priority Interrupt. Specifies the 7-bit unique interrupt number/vector (Table 20-4) of the single interrupt controller interrupt source advanced to the highest priority in the IPIC Priority table (Table 20-27). HPI can be modified dynamically.
MPSB	Mixed interrupts Priority Scheme for group B. Selects the relative MIXB priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXBs are grouped by priority at the top of the table. 1 Spread. The MIXBs are spread by priority in the table.
MPSA	Mixed interrupts Priority Scheme for group A. Selects the relative MIXA priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXAs are grouped by priority at the top of the table. 1 Spread. The MIXAs are spread by priority in the table.
IPSD	Internal interrupts Priority Scheme for group D. Selects the relative SYSD priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSDs are grouped by priority at the top of the table. 1 Spread. The SYSDs are spread by priority in the table.

Offset 0x00Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	HPI							0	MPSB	MPSA	0	IPSD	IPSC	IPSB	IPSA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	HPIT		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-2. System Global Interrupt Configuration Register (SICFR)
(The register is repeated for reference.)

Table 20-2. SICFR Field Descriptions (continued)

Field	Description
IPSC	Internal interrupts Priority Scheme for group C. Selects the relative SYSC priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSCs are grouped by priority at the top of the table. 1 Spread. The SYSCs are spread by priority in the table.
IPSB	Internal interrupts Priority Scheme for group B. Selects the relative SYSB priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSBs are grouped by priority at the top of the table. 1 Spread. The SYSBs are spread by priority in the table.
IPSA	Internal Interrupts Priority Scheme for Group A. Selects the relative SYSA priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSAs are grouped by priority at the top of the table. 1 Spread. The SYSAs are spread by priority in the table.
HPIT	HPI Priority Position IPIC Output Interrupt Type. Defines type of the IPIC output interrupt signal (\overline{int} , \overline{cint} , or \overline{smi}) asserts its request to the core in the HPI priority position. These bits cannot be changed dynamically. If S/W really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change. The definition of HPIT is as follows: 00 \overline{int} request is asserted to the core for HPI. 01 \overline{smi} request is asserted to the core for HPI. 10 \overline{cint} request is asserted to the core for HPI. 11 Reserved.

20.2.1.2 System Global Interrupt Vector Register (SIVCR)

SIVCR, shown in Figure 20-3, contains a 7-bit code (Table 20-3) representing the unmasked interrupt source of the highest priority level.

Offset 0x04 Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	IVEC						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-3. System Global Interrupt Vector Register (SIVCR)

Table 20-3. SIVCR Field Descriptions

Field	Description
IVEC	<p>Interrupt Vector. Specifies a 7-bit unique number of the IPIC's highest priority interrupt source, pending to the core. When an interrupt request occurs, SIVCR can be read. If there are multiple interrupt sources, SIVCR latches the highest priority interrupt.</p> <p>Note: The value of SIVEC cannot change while it is being read.</p>

Table 20-4 shows the definition of IVEC.

NOTE

Interrupt vector numbers are assigned to each module and cannot be changed. For example, the PCI module always returns 0x000.0001 as its interrupt vector regardless of its relative priority in the SYSC group.

Table 20-4. IVEC Field Values (Sheet 1 of 3)

Interrupt Number	Meaning	Interrupt Vector	Default Group Programing
0	Error (No Interrupt)	0x000_0000	—
1	PCI	0x000_0001	SYSC0(Grouped)
2	PCI DMA	0x000_0010	SYSC1(Grouped)
3	PCI MU	0x000_0011	SYSC2(Grouped)
4	FEC	0x000_0100	SYSC3(Grouped)
5	PATA	0x000_0101	SYSC4(Grouped)

Table 20-4. IVEC Field Values (Sheet 2 of 3)

Interrupt Number	Meaning	Interrupt Vector	Default Group Programing
6	NFC	0x000_0110	SYSC5(Grouped)
7	LPC	0x000_0111	SYSC6(Grouped)
8	SDHC	0x000_1000	SYSC7(Grouped)
9	I2C1	0x000_1001	SYSD0(Grouped)
10	I2C2	0x000_1010	SYSD1(Grouped)
11	I2C3	0x000_1011	SYSD2(Grouped)
12	MSCAN1	0x000_1100	SYSD3(Grouped)
13	MSCAN2	0x000_1101	SYSD4(Grouped)
14	BDLC	0x000_1110	SYSD5(Grouped)
15	GPT0	0x000_1111	SYSD6(Grouped)
16	GPT1	0x001_0000	SYSD7(Grouped)
17	ipp_ind_ext_int[1]	0x001_0001	MIXA5(Grouped)
18	ipp_ind_ext_int[2]	0x001_0010	MIXA6(Grouped)
19	ipp_ind_ext_int[3]	0x001_0011	MIXA7(Grouped)
20	Reserved	0x001_0100	MIXB4(Grouped)
21	Reserved	0x001_0101	MIXB5(Grouped)
22	Reserved	0x001_0110	MIXB6(Grouped)
23	Reserved	0x001_0111	MIXB7(Grouped)
24	Reserved	0x001_1000	Fixed priority
25	Reserved	0x001_1001	Fixed priority
26	Reserved	0x001_1010	Fixed priority
27	Reserved	0x001_1011	Fixed priority
28	Reserved	0x001_1100	Fixed priority
29	Reserved	0x001_1101	Fixed priority
30	Reserved	0x001_1110	Fixed priority
31	Reserved	0x001_1111	Fixed priority
32	PSC4	0x010_0000	SYSA0(Grouped)
33	PSC5	0x010_0001	SYSA1(Grouped)
34	PSC6	0x010_0010	SYSA2(Grouped)
35	PSC7	0x010_0011	SYSA3(Grouped)
36	PSC8	0x010_0100	SYSA4(Grouped)
37	PSC9	0x010_0101	SYSA5(Grouped)
38	PSC10	0x010_0110	SYSA6(Grouped)
39	PSC11	0x010_0111	SYSA7(Grouped)
40	FIFOC	0x010_1000	SYSB0(Grouped)
41	SPDIF	0x010_1001	SYSB1(Grouped)
42	AXE	0x010_1010	SYSB2(Grouped)
43	USB ULPI	0x010_1011	SYSB3(Grouped)
44	USB UTMI	0x010_1100	SYSB4(Grouped)

Table 20-4. IVEC Field Values (Sheet 3 of 3)

Interrupt Number	Meaning	Interrupt Vector	Default Group Programing
45	SATA	0x010_1101	SYSB5(Grouped)
46	Reserved	0x010_1110	SYSB6(Grouped)
47	Reserved	0x010_1111	SYSB7(Grouped)
48	ipp_ind_ext_int[0]	0x011_0000	MIXA4(Grouped)
49–63	Reserved	0x011_0001 – 0x011_1111	Reserved
64	ipi_int_internal[32]	0x100_0000	MIXA0(Grouped)
65	ipi_int_internal[33]	0x100_0001	MIXA1(Grouped)
66	ipi_int_internal[34] ¹	0x100_0010	MIXA2(Grouped)
67	ipi_int_internal[35]	0x100_0011	MIXA3(Grouped)
68	PSC0	0x100_0100	MIXB0(Grouped)
69	PSC1	0x100_0101	MIXB1(Grouped)
70	PSC2	0x100_0110	MIXB2(Grouped)
71	PSC3	0x100_0111	MIXB3(Grouped)
72	GPT2	0x100_1000	Fixed priority
73	GPT3	0x100_1001	Fixed priority
74	GPT4	0x100_1010	Fixed priority
75	GPT5	0x100_1011	Fixed priority
76	GPT6	0x100_1100	Fixed priority
77	GPT7	0x100_1101	Fixed priority
78	GPIO	0x100_1110	Fixed priority
79	RTC SEC	0x100_1111	Fixed priority
80	RTC ALARM	0x101_0000	Fixed priority
81	DDR	0x101_0001	Fixed priority
82	SBA	0x101_0010	Fixed priority
83	PMC	0x101_0011	Fixed priority
84	USB ULPI WKUP	0x101_0100	Fixed priority
85	USB UTMI WKUP	0x101_0101	Fixed priority
86	SATA CMD	0x101_0110	Fixed priority
87	TEMP 105C	0x101_0111	Fixed priority
88	IIM	0x101_1000	Fixed priority
89	DDR PRIOMAN	0x101_1001	Fixed priority
90	MSCAN3	0x101_1010	Fixed priority
91	MSCAN4	0x101_1011	Fixed priority
92	Reserved	0x101_1100	Fixed priority
93	Reserved	0x101_1101	Fixed priority
95	Reserved	0x101_1110	Fixed priority
95	Reserved	0x101_1111	Fixed priority

¹ Not available in MPC5123. Therefore, it is reserved.

20.2.1.3 System Internal Interrupt Pending Registers (SIPNR_H and SIPNR_L)

Each bit in the system internal interrupt pending registers (SIPNR_H and SIPNR_L), shown in [Figure 20-4](#) and [Figure 20-5](#), corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is managed, clear the SIPNR bit by clearing the corresponding event register bit.

Offset 0x08 Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PSC4	PSC5	PSC6	PSC7	PSC8	PSC9	PSC10	PSC11	FIFOC	SPDIF	AXE	USB ULPI	USB UTMI	SATA		
W															0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PCI	PCI DMA	PCI MU	FEC	PATA	NFC	LPC	SDHC	I2C1	I2C2	I2C3	MSCAN1	MSCAN2	BDLC	GPT0	GPT1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-4. System Internal Interrupt Pending Register (SIPNR_H)

Table 20-5. SIPNR_H Field Descriptions

Field	Description
[31:0]	Each bit corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is managed, clear the corresponding SIPNR bit. However, if an event register exists, the unmasked event register bits should be cleared instead, causing the SIPNR bit to be cleared. SIPNR bits are read only. Writing to this register has no effect. Note: The SIPNR bit positions are not changed according to their relative priority.

Integrated Programmable Interrupt Controller (IPIC)

Offset 0x0C Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipi_int_inter nal[32]	ipi_int_inter nal[33]	ipi_int_inter nal[34]	ipi_int_inter nal[35]	PSC0	PSC1	PSC2	PSC3	GPT2	GPT3	GPT4	GPT5	GPT6	GPT7	GPIO	RTC SEC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RTC ALARM	DDR	SBA	PMC	USB ULPI WKUP	USB UTMI WKUP	SATA CMD	TEMP 105C	IIM	DDR PRI- OMA N	MSCA N3	MSCA N4				
W													0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-5. System Internal Interrupt Pending Register (SIPNR_L)

Table 20-6. SIPNR_L Field Descriptions

Field	Description
[31:0]	<p>Each bit corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is managed, clear the corresponding SIPNR bit. However, if an event register exists, the unmasked event register bits should be cleared instead, causing the SIPNR bit to be cleared.</p> <p>SIPNR bits are read only. Writing to this register has no effect.</p> <p>Note: The SIPNR bit positions are not changed according to their relative priority.</p>

¹ Not available in MPC5123.

20.2.1.4 System Internal Interrupt Group A Priority Register (SIPRR_A)

The system internal interrupt group A priority register (SIPRR_A), shown in Figure 20-6, defines the priority between PSC4, PSC5, PSC6, PSC7, PSC8, PSC9, PSC10, and PSC11 internal interrupt signals.

Offset 0x10 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSA0P				SYSA1P				SYSA2P				SYSA3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSA4P				SYSA5P				SYSA6P				SYSA7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-6. System Internal Interrupt Group A Priority Register (SIPRR_A)

Table 20-7. SIPRR_A Field Descriptions

Field	Description
SYSA0P	SYSA0 Priority Order. Defines which interrupt source asserts its request in the SYSA0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSA0P is shown as follows: 000 PSC4 asserts its request in the SYSA0 position. 001 PSC5 asserts its request in the SYSA0 position. 010 PSC6 asserts its request in the SYSA0 position. 011 PSC7 asserts its request in the SYSA0 position. 100 PSC8 asserts its request in the SYSA0 position. 101 PSC9 asserts its request in the SYSA0 position. 110 PSC10 asserts its request in the SYSA0 position. 111 PSC11 asserts its request in the SYSA0 position.
SYSA1P–SYSA7P	Same as SYSA0P, but for SYSA1P–SYSA7P.

20.2.1.5 System Internal Interrupt Group B Priority Register (SIPRR_B)

The System Internal Interrupt Group B Priority Register (SIPRR_B), shown in Figure 20-7, defines the priority between FIFOC, SPDIF, AXE, USB ULPI, USB UTMI, SATA, Reserved, and Reserved internal interrupt signals.

Offset 0x14 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSB0P				SYSB1P				SYSB2P				SYSB3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSA4P				SYSA5P				SYSB6P				SYSB7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-7. System Internal Interrupt Group B Priority Register (SIPRR_B)

Table 20-8. SIPRR_B Field Descriptions

Field	Description
SYSB0P	SYSB0 Priority order. Defines which interrupt source asserts its request in the SYSB0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSB0P is shown as follows: 000 FIFOC asserts its request in the SYSB0 position. 001 SPDIF asserts its request in the SYSB0 position. 010 AXE asserts its request in the SYSB0 position. 011 USB ULPI asserts its request in the SYSB0 position. 100 USB UTMI asserts its request in the SYSB0 position. 101 SATA asserts its request in the SYSB0 position. 110 Reserved asserts its request in the SYSB0 position. 111 Reserved asserts its request in the SYSB0 position.
SYSB1P–SYSB7P	Same as SYSB0P, but for SYSB1P–SYSB7P.

20.2.1.6 System Internal Interrupt Group C Priority Register (SIPRR_C)

The System Internal Interrupt Group C Priority Register (SIPRR_C), shown in Figure 20-8, defines the priority between PCI, PCI DMA, PCI MU, FEC, PATA, NFC, LPC, and SDHC internal interrupt signals.

Offset 0x18 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSC0P				SYSC1P				SYSC2P				SYSC3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSC4P				SYSC5P				SYSC6P				SYSC7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-8. System Internal Interrupt Group C Priority Register (SIPRR_C)

Table 20-9. SIPRR_C Field Descriptions

Field	Description
SYSC0P	SYSC0 Priority order. Defines which interrupt source asserts its request in the SYSC0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSC0P is shown as follows: 000 PCI asserts its request in the SYSC0 position. 001 PCI DMA asserts its request in the SYSC0 position. 010 PCI MU asserts its request in the SYSC0 position. 011 FEC asserts its request in the SYSC0 position. 100 PATA asserts its request in the SYSC0 position. 101 NFC asserts its request in the SYSC0 position. 110 LPC asserts its request in the SYSC0 position. 111 SDHC asserts its request in the SYSC0 position.
SYSC1P–SYSC7P	Same as SYSC0P, but for SYSC1P–SYSC7P.

20.2.1.7 System Internal Interrupt Group D Priority Register (SIPRR_D)

The system internal interrupt group D priority register (SIPRR_D), shown in Figure 20-9, defines the priority between I2C1, I2C2, I2C3, MSCAN1, MSCAN2, BDLC, GPT0, and GPT1 internal interrupt signals.

Offset 0x1C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSD0P				SYSD1P				SYSD2P				SYSD3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSD4P				SYSD5P				SYSD6P				SYSD7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-9. System Internal Interrupt Group D Priority Register (SIPRR_D)

Table 20-10. SIPRR_D Field Descriptions

Field	Description
SYSD0P	SYSD0 Priority order. Defines which interrupt source asserts its request in the SYSD0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSD0P is shown as follows: 000 I2C1 asserts its request in the SYSD0 position. 001 I2C2 asserts its request in the SYSD0 position. 010 I2C3 asserts its request in the SYSD0 position. 011 MSCAN1 asserts its request in the SYSD0 position. 100 MSCAN2 asserts its request in the SYSD0 position. 101 BDLC asserts its request in the SYSD0 position. 110 GPT0 asserts its request in the SYSD0 position. 111 GPT1 asserts its request in the SYSD0 position.
SYSD1P–SYSD7P	Same as SYSD0P, but for SYSD1P–SYSD7P.

20.2.1.8 System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)

Each implemented bit in the SIMSR_H and SIMSR_L, shown in [Figure 20-10](#) and [Figure 20-11](#), corresponds to an internal interrupt source. Mask an interrupt by setting the corresponding SIMSR bit. When an interrupt request occurs, the corresponding SIPNR bit is set, regardless of the SIMSR bit. However, if the corresponding SIMSR bit is cleared, no interrupt request is passed to the core.

When the SIMSR bit is cleared at the same time an interrupt source requests an interrupt service, the request stops. If you set the SIMSR bit later, a previously pending interrupt request is processed by the core according to its assigned priority.

Offset 0x20 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PSC4	PSC5	PSC6	PSC7	PSC8	PSC9	PSC10	PSC11	FIFOC	SPDIF	AXE	USB ULPI	USB UTMI	SATA	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PCI	PCI DMA	PCI MU	FEC	PATA	NFC	LPC	SDHC	I2C1	I2C2	I2C3	MSC AN1	MSC AN2	BDLC	GPT0	GPT1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-10. System Internal Interrupt Mask Register (SIMSR_H)

Table 20-11. SIMSR_H Field Descriptions

Field	Description
Refer to Figure 20-10	<p>Each bit corresponds to an external interrupt source. Mask an interrupt by clearing the SIMSR bit. An interrupt can be enabled by setting the corresponding SIMSR bit. The SIMSR can be read at any time.</p> <p>Note:</p> <ul style="list-style-type: none"> SIMSR bit positions are not changed according to their relative priority. You can clear pending register bits set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. If an SIMSR bit is masked at the same time the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Therefore, always include an error.

Integrated Programmable Interrupt Controller (IPIC)

Offset 0x24Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipi_int_inter- nal[32]	ipi_int_inter- nal[33]	ipi_int_inter- nal[34] ¹	ipi_int_inter- nal[35]	PSC0	PSC1	PSC2	PSC3	GPT2	GPT3	GPT4	GPT5	GPT6	GPT7	GPIO	RTC SEC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RTC ALARM	DDR	SBA	PMC	USB ULPI WKUP	USB UTMI WKUP	SATA CMD	TEMP 105C	IIM	DDR PRI- OMAN	MSCA N3	MSCA N4	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-11. System Internal Interrupt Mask Register (SIMSR_L)

Table 20-12. SIMSR_L Field Descriptions

Field	Description
[31:0]	<p>Each bit corresponds to an external interrupt source. Mask an interrupt by clearing the SIMSR bit. An interrupt can be enabled by setting the corresponding SIMSR bit. The SIMSR can be read at any time.</p> <p>Note:</p> <ul style="list-style-type: none"> SIMSR bit positions are not changed according to their relative priority. You can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. If an SIMSR bit is masked at the same time the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Therefore, always include an error.

¹ Not available in MPC5123

20.2.1.9 System Internal Interrupt Control Register (SICNR)

SICNR, shown in Figure 20-12, defines the IPIC output interrupt type (\overline{int} , \overline{cint} , or \overline{smi}) in the SYSA0-SYSA1, SYSB0-SYSB1, SYSC0-SYSC1, and SYSD0-SYSD1 priority positions. All other priority positions assert an \overline{INT} signal to the core.

Offset 0x28 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSD0T		SYSD1T		0	0	0	0	SYSC0T		SYSC1T		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSB0T		SYSB1T		0	0	0	0	SYSA0T		SYSA1T		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-12. System Internal Interrupt Control Register (SICNR)
(Register is repeated for reference.)

Table 20-13. SICNR Field Descriptions

Field	Description
SYSDxT	<p>SYSDx Priority Position IPIC Output Interrupt Type. Defines which type of IPIC output interrupt (\overline{int}, \overline{cint}, or \overline{smi}) asserts its request to the core in the SYSDx priority position. These bits cannot be changed dynamically. To change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change.</p> <p>The definition of SYSDxT is as follows:</p> <p>00 \overline{int} request is asserted to the core for SYSDx.</p> <p>01 \overline{smi} request is asserted to the core for SYSDx.</p> <p>10 \overline{cint} request is asserted to the core for SYSDx.</p> <p>11 Reserved</p>
SYSCxT	<p>SYSCx Priority Position IPIC Output Interrupt Type. Defines which type of IPIC output interrupt (\overline{int}, \overline{cint}, or \overline{smi}) asserts its request to the core in the SYSCx priority position. These bits cannot be changed dynamically. To change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change.</p> <p>The definition of SYSCxT is as follows:</p> <p>00 \overline{int} request is asserted to the core for SYSCx.</p> <p>01 \overline{smi} request is asserted to the core for SYSCx.</p> <p>10 \overline{cint} request is asserted to the core for SYSCx.</p> <p>11 Reserved</p>

Integrated Programmable Interrupt Controller (IPIC)

Offset 0x28Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYSD0T		SYSD1T		0	0	0	0	SYSC0T		SYSC1T		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SYSB0T		SYSB1T		0	0	0	0	SYSA0T		SYSA1T		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-12. System Internal Interrupt Control Register (SICNR)
(Register is repeated for reference.)

Table 20-13. SICNR Field Descriptions (continued)

Field	Description
SYSBxT	<p>SYSBx Priority Position IPIC Output Interrupt Type. Defines which type of IPIC output interrupt (\overline{int}, \overline{cint}, or \overline{smi}) asserts its request to the core in the SYSBx priority position. These bits cannot be changed dynamically. To change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change.</p> <p>The definition of SYSBxT is as follows:</p> <p>00 \overline{int} request is asserted to the core for SYSBx.</p> <p>01 \overline{smi} request is asserted to the core for SYSBx.</p> <p>10 \overline{cint} request is asserted to the core for SYSBx.</p> <p>11 Reserved</p>
SYSAxT	<p>SYSAx Priority Position IPIC Output Interrupt Type. Defines which type of IPIC output interrupt (\overline{int}, \overline{cint}, or \overline{smi}) asserts its request to the core in the SYSAx priority position. These bits cannot be changed dynamically. To change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change.</p> <p>The definition of SYSAxT is as follows:</p> <p>00 \overline{int} request is asserted to the core for SYSAx.</p> <p>01 \overline{smi} request is asserted to the core for SYSAx.</p> <p>10 \overline{cint} request is asserted to the core for SYSAx.</p> <p>11 Reserved</p>

20.2.1.10 System External Interrupt Pending Register (SEPNR)

Each bit in the SEPNR, shown in Figure 20-13, corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.

Offset 0x2C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_ind_ext_int[0] ¹	ipp_ind_ext_int[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	Note ²	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 20-13. System External Interrupt Pending Register (SEPNR)

Table 20-14. SEPNR Field Descriptions

Field	Description
[31:0]	<p>Each bit corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.</p> <p>When a pending interrupt is managed, clear the corresponding SEPNR bit. For level triggered case, s/w needs to negate the $\overline{\text{IRQx}}$ that automatically clears the bit in SEPNR, and for edge triggered case, s/w needs to clear SEPNR.</p> <p>SEPNR bits are cleared by writing ones to them. Because the user can only clear bits in this register, writing zeros to this register has no effect.</p> <p>Note: The SEPNR bit positions are not changed according to their relative priority.</p>

¹ This bit is valid only if the ipp_ind_ext_int[0] signal is configured as an external maskable interrupt (SEMSR[Sipp_ind_ext_int[0]] = 0)

² Reflect the state of external IRQ# pins. User should take care to drive all $\overline{\text{IRQ}}$ inputs to inactive state prior to reset negation

20.2.1.11 System Mixed Interrupt Group A Priority Register (SMPRR_A)

The SMPRR_A, shown in Figure 20-14, defines the priority between ipi_int_internal[32], ipi_int_internal[33], ipi_int_internal[34], ipi_int_internal[35], ipp_ind_ext_int[0], ipp_ind_ext_int[1], ipp_ind_ext_int[2], and ipp_ind_ext_int[3].

Offset 0x30 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MIXA0P				MIXA1P				MIXA2P				MIXA3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MIXA4P				MIXA5P				MIXA6P				MIXA7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-14. System Mixed Interrupt Group A Priority Register (SMPRR_A)

Table 20-15. SMPRR_A Field Descriptions

Field	Description
MIXA0P	<p>MIXA0 Priority order. Defines which interrupt source asserts its request in the MIXA0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXA0P is as follows:</p> <p>000 ipi_int_internal[32] asserts its request to the MIXA0 position.</p> <p>001 ipi_int_internal[33] asserts its request to the MIXA0 position.</p> <p>010 ipi_int_internal[34] asserts its request to the MIXA0 position.</p> <p>011 ipi_int_internal[35] asserts its request to the MIXA0 position.</p> <p>100 ipp_ind_ext_int[0] asserts its request to the MIXA0 position. This field for MIXA0 position is valid (must not be ignored) if ipp_ind_ext_int[0] signal configured as an external maskable interrupt (SEMSR[Sipp_ind_ext_int[0]] = 0).</p> <p>101 ipp_ind_ext_int[1] asserts its request to the MIXA0 position.</p> <p>110 ipp_ind_ext_int[2] asserts its request to the MIXA0 position.</p> <p>111 ipp_ind_ext_int[3] asserts its request to the MIXA0 position.</p>
MIXA1P–MIXA7P	Same as MIXA0P, but for MIXA1P–MIXA7P.

20.2.1.12 System Mixed Interrupt Group B Priority Register (SMPRR_B)

The system mixed interrupt group B priority register (SMPRR_B), shown in Figure 20-15, defines the priority between PSC0, PSC1, PSC2, PSC3, Reserved, Reserved, Reserved, and Reserved.

Offset 0x34 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MIXB0P				MIXB1P				MIXB2P				MIXB3P			
W																
Reset	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MIXB4P				MIXB5P				MIXB6P				MIXB7P			
W																
Reset	1	0	0	1	0	1	1	1	0	1	1	1	0	0	0	0

= Unimplemented or Reserved

Figure 20-15. System Mixed Interrupt Group B Priority Register (SMPRR_B)

Table 20-16. SMPRR_B Field Descriptions

Field	Description
MIXB0P	<p>MIXB0 Priority order. Defines which interrupt source asserts its request in the MIXB0 priority position. Do not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXB0P is as follows:</p> <p>000 PSC0 asserts its request to the MIXB0 position.</p> <p>001 PSC1 asserts its request to the MIXB0 position.</p> <p>010 PSC2 asserts its request to the MIXB0 position.</p> <p>011 PSC3 asserts its request to the MIXB0 position.</p> <p>100 Reserved asserts its request to the MIXB0 position.</p> <p>101 Reserved asserts its request to the MIXB0 position.</p> <p>110 Reserved asserts its request to the MIXB0 position.</p> <p>111 Reserved asserts its request to the MIXB0 position.</p>
MIXB1P–MIXB7P	Same as MIXB0P, but for MIXB1P–MIXB7P.

20.2.1.13 System External Interrupt Mask Register (SEMSR)

Each bit in the SEMSR, shown in Figure 20-16, corresponds to an external interrupt source. Mask an interrupt by clearing the corresponding SEMSR bit. An interrupt is unmasked (enabled) by setting the SEMSR bit.

When an external interrupt request occurs, the corresponding SEP NR bit is set regardless of the SEMSR bit. However, if the corresponding SEMSR bit is cleared, no interrupt request is passed to the core.

When the SEMSR bit is cleared at the same time an interrupt source requests an interrupt service, the request stops. If you set the SEMSR bit later, a previously pending interrupt request is processed by the core according to its assigned priority. The SEMSR can be read at any time.

Offset 0x38 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_ind_ext_int[0] ¹	ipp_ind_ext_int[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Sipp_ind_ext_int[0]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 20-16. System External Interrupt Mask Register (SEMSR)

Table 20-17. SEMSR Field Descriptions

Field	Description
Refer to Figure 20-16	Each bit corresponds to an external interrupt source. Mask an interrupt by clearing the SEMSR bit. An interrupt can be enabled by setting the corresponding SEMSR bit. The SEMSR can be read by the user at any time. Note: <ul style="list-style-type: none"> SEMSR bit positions are not affected by their relative priority. You can clear pending register bits set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. If an SEMSR bit is masked at the same time the corresponding SEP NR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Thus, you must always include an error vector routine, even if it contains only an RFI instruction. The error vector cannot be masked.
SIRQ0	Steer ipp_ind_ext_int[0]. 0 ipp_ind_ext_int[0] is used as external interrupt request 1 ipp_ind_ext_int[0] is used as external MCP request

¹ This bit is valid only if the ipp_ind_ext_int[0] signal is configured as an external maskable interrupt (SEMSR[Sipp_ind_ext_int[0]] = 0)

20.2.1.14 System External Interrupt Control Register (SECNR)

The SECNR, shown in Figure 20-17, defines the edge detect mode for external $\overline{\text{IRQ}}$ interrupt signals, determines whether the corresponding $\overline{\text{IRQx}}$ signal asserts an interrupt request upon either a high-to-low change or low assertion on the pin. It also defines the IPIC output interrupt type (*int*, *cint*, or *smi*) in the MIXA0-MIXA1 and MIXB0-MIXB1 priority positions.

Offset 0x3C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MIXB0T		MIXB1T		0	0	0	0	MIXA0T		MIXA1T		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EDIO	EDI1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-17. System External Interrupt Control Register (SECNR)
(The register is repeated for reference.)

Table 20-18. SECNR Field Descriptions

Field	Description
MIXB0T	MIXB0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (<i>int</i> , <i>cint</i> , or <i>smi</i>) asserts its request to the core in the MIXB0 priority position. These bits can be changed dynamically. The definition of MIXB0T is as follows: 00 <i>int</i> request is asserted to the core for MIXB0. 01 <i>smi</i> request is asserted to the core for MIXB0. 10 <i>cint</i> request is asserted to the core for MIXB0. 11 Reserved
MIXB1T	Same as MIXB0T, but for MIXB1T.
MIXA0T	MIXA0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal (<i>int</i> , <i>cint</i> , or <i>smi</i>) asserts its request to the core in the MIXB0 priority position. These bits can be changed dynamically. The definition of MIXA0T is as follows: 00 <i>int</i> request is asserted to the core for MIXA0. 01 <i>smi</i> request is asserted to the core for MIXA0. 10 <i>cint</i> request is asserted to the core for MIXA0. 11 Reserved
MIXA1T	Same as MIXA0T, but for MIXA1T.
EDix	Each bit defines the edge detect mode for the external $\overline{\text{IRQ}}$ interrupt signals, and determines whether the corresponding $\overline{\text{IRQx}}$ signal asserts an interrupt request upon either a high-to-low change or low assertion on the pin. The corresponding $\overline{\text{IRQx}}$ signal asserts an interrupt request as follows: 0 Low assertion on $\overline{\text{IRQx}}$ generates an interrupt request. 1 High-to-low change on $\overline{\text{IRQx}}$ generates an interrupt request.

20.2.1.15 System Error Status Register (SERSR)

Each bit in the SERSR, shown in [Figure 20-18](#), corresponds to an external and an internal non-maskable error source machine check (*mcp*). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit.

Offset 0x40 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_in	WDT	SBA	0	0	PCI	0	MU	0	TEMP	0	0	0	0	0	0
W	d_ext_int[0]								w1c	125C						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-18. System Error Status Register (SERSR)

Table 20-19. SERSR Field Descriptions

Field	Description
Refer to Figure 20-18	Each bit in the system error status register (SERSR), shown in Figure 20-18 , corresponds to an external and an internal error source (MCP). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit. SERSR bits are cleared by writing ones to them. The unmasked event register bits should be cleared before clearing of SERSR. Because you can only clear bits in this register, writing zeros to this register has no effect. SERSR bits are reset only by power on reset. Soft and hard reset are not affected SERSR bit states.

20.2.1.16 System Error Mask Register (SERMR)

Each bit in SERMR, shown in [Figure 20-19](#), corresponds to an external and an internal \overline{mcp} source. Mask a MCP by clearing and enables a MCP by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the SERMR bit although no MCP request is passed to the core. The SERMR can be read by the user at any time.

Offset 0x44Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_in			0	0		0				0	0	0	0	0	0
W	d_ext_int[0]	WDT	SBA			PCI		MU	1	TEMP 125C						
Reset	1	1	1	0	0	1	0	1	1	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-19. System Error Mask Register (SERMR)

Table 20-20. SERMR Field Descriptions

Field	Description
Refer to Figure 20-19	Each bit in the System Error Status Register (SERMR), shown in Figure 20-19 , corresponds to an external and an internal MCP source. Mask an MCP by clearing and enables a MCP by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the SERMR bit, although no MCP request is passed to the core. The SERMR can be read by the user at any time.

20.2.1.17 System Internal Interrupt Force Register (SIFCR_H and SIFCR_L)

Each bit in SIFCR_H and SIFCR_L, shown in [Figure 20-20](#) and [Figure 20-21](#), corresponds to an internal interrupt source. When a bit is set, the interrupt controller generate the corresponding interrupt (sets the corresponding SIPNR bit). The SIFCR can be read by the user at any time.

NOTE

If an internal interrupt is generated by setting the corresponding SIFCR bit, the corresponding SIPNR bit is set until the corresponding SIFCR bit is cleared.

Offset 0x50 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PSC4	PSC5	PSC6	PSC7	PSC8	PSC9	PSC10	PSC11	FIFOC	SP-DIF	AXE	USB ULPI	USB UTMI	SATA	0	0
W																
Reset																
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PCI	PCI DMA	PCI MU	FEC	PATA	NFC	LPC	SDHC	I2C1	I2C2	I2C3	MSC AN1	MSC AN2	BDLC	GPT0	GPT1
W																
Reset																


 = Unimplemented or Reserved

Figure 20-20. System Internal Interrupt Force Register (SIFCR_H)

Table 20-21. SIFCR_H Field Descriptions

Field	Description
Refer to Figure 20-20	Each bit corresponds to an interrupt source. Force an interrupt by setting the SIFCR bit. An interrupt can be enabled by setting the corresponding SIFCR bit. Note: SIFCR bit positions are not changed according to their relative priority.

Integrated Programmable Interrupt Controller (IPIC)

Offset 0x54Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipi_int_inter- nal[32]	ipi_int_inter- nal[33]	ipi_int_inter- nal[34]	ipi_int_inter- nal[35]	PSC0	PSC1	PSC2	PSC3	GPT2	GPT3	GPT4	GPT5	GPT6	GPT7	GPIO	RTC SEC
W																
Reset																
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RTC ALARM	DDR	SBA	PMC	USB ULPI WKUP	USB UTMI WKUP	SATA CMD	TEMP 105C	IIM	DDR PRI- OMAN	MSCA N3	MSCA N4	0	0	0	0
W																
Reset																

= Unimplemented or Reserved

Figure 20-21. System Internal Interrupt Force Register (SIFCR_L)

Table 20-22. SIFCR_L Field Descriptions

Field	Description
Refer to Figure 20-21	Each bit corresponds to an interrupt source. Force an interrupt by setting the SIFCR bit. An interrupt can be enabled by setting the corresponding SIFCR bit. Note: SIFCR bit positions are not changed according to their relative priority.

¹ Not available in MPC5123

20.2.1.18 System External Interrupt Force Register (SEFCR)

Each bit in SEFCR, shown in Figure 20-22, corresponds to an external interrupt source. When a bit is set, the interrupt controller generates the corresponding external interrupt (sets the corresponding SEPNR bit).

The SEFCR can be read by the user at any time.

NOTE

If an external interrupt is configured to operate in level-sensitive mode and generated by setting the SEFCR bit, the corresponding SEPNR bit is set until the corresponding SEFCR bit is cleared.

Offset 0x58 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_ind_ext_int[0] ¹	ipp_ind_ext_int[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0														
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset																

= Unimplemented or Reserved

Figure 20-22. System External Interrupt Force Register (SEFCR)

Table 20-23. SEFCR Field Descriptions

Field	Description
IRQ0–IRQ1	Each bit corresponds to an external interrupt source. Force an interrupt by setting the SIFCR bit. An interrupt can be enabled by setting the corresponding SIFCR bit. Note: SIFCR bit positions are not changed according to their relative priority.

¹ This bit is valid only if the ipp_ind_ext_int[0] signal is configured as an external maskable interrupt (SEMSR[Sipp_ind_ext_int[0]] = 0)

20.2.1.19 System Error Force Register (SERFR)

Each bit in SERFR, shown in Figure 20-23, corresponds to an external MCP source. When a bit is set, the interrupt controller generates the corresponding MCP interrupt (sets the corresponding SERSR bit).

The SERFR can be read by the user at any time.

Offset 0x5C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ipp_ind_ext_int[0] ₁	WDT	SBA	0	0	PCI	0	MU	Reserved	TEMP 125C	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

1. This bit is valid only if the ipp_ind_ext_int[0] signal is configured as an external MCP interrupt (SEMSR[Sipp_ind_ext_int[0]] = 1)

Figure 20-23. System Error Force Register (SERFR)

Table 20-24. SERFR Field Descriptions

Field	Description
Refer to Figure 20-23	Each bit corresponds to an external MCP source. You can force an MCP by setting the SERFR bit. Note: SERFR bit positions are not affected by their relative priority.

20.2.1.20 System Critical Interrupt Vector Register (SCVCR)

SCVCR, shown in [Figure 20-24](#), contains a 7-bit code ([Table 20-25](#)) representing the unmasked critical interrupt (*cint*) source of the highest priority level.

Offset 0x60 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	CVEC						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-24. System Critical Interrupt Vector Register (SCVCR)

Table 20-25. SCVCR Field Descriptions

Field	Description
CVEC	Critical interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority critical interrupt source, pending to the core. When a critical interrupt request occurs, SCVCR can be read. If there are multiple critical interrupt sources, SCVCR latches the highest priority critical interrupt. The CVEC field correctly reflects all of the interrupt vectors (See Table 20-4 for details). The value of SCVEC cannot change while it is being read.

20.2.1.21 System Management Interrupt Vector Register (SMVCR)

SMVCR, shown in [Figure 20-25](#), contains a 7-bit code ([Table 20-26](#)) representing the unmasked system management interrupt (SMI) source of the highest priority level.

Offset 0x64 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	MVEC						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 20-25. System Management Interrupt Vector Register (SMVCR)

Table 20-26. SMVCR Field Descriptions

Field	Description
MVEC	System management interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority system management interrupt source, pending to the core. When a system management interrupt request occurs, SMVCR can be read. If there are multiple system management interrupt sources, SMVCR latches the highest priority system management interrupt. The MVEC field correctly reflects all interrupt vectors (See Table 20-4 for details). The value of SMVEC cannot change while it is being read.

20.3 Functional Description

The following sections describe the type of interrupts, interrupt configurations, and their priorities.

20.3.1 Interrupt Types

The IPIC is responsible for receiving hardware-generated interrupts from different sources (both internal and external) along with prioritizing and delivering them to the CPU for servicing. The interrupt sources are controlled by the IPIC unit and may cause three types of exceptions in the processor core. The *int* signal is the main interrupt output from the IPIC to the processor core and causes the external interrupt exception. The *cint* signal is the critical interrupt output from the IPIC to the processor core and causes the critical external interrupt exception. The *smi* signal is the system management interrupt output from the IPIC to the processor core and causes the system management interrupt exception. The machine check exception is caused by the internal *mcp* signal generated by the IPIC, informing the processor of error conditions, assertion of the external MCP request, and other conditions.

20.3.2 Interrupt Configuration

Figure 20-26 shows the interrupt configuration of the MPC5121e.

The interrupt controller allows masking of each interrupt source. When an unmasked interrupt source is pending in the SIPNR register, the interrupt controller sends an interrupt request to the core. When an interrupt is taken, the interrupt mask bit in the machine state register is cleared to disable further interrupt requests to the Power Architecture core until software can manage them.

All interrupt sources are prioritized and bits are set in the system interrupt pending register (SIPNR, SEP NR) as interrupts occur regardless of whether they are masked in the IPIC. The prioritization of the interrupt sources is flexible within the following groups:

- The relative priority of the PSC4, PSC5, PSC6, PSC7, PSC8, PSC9, PSC10, and PSC11 internal interrupt signals can be modified.
- The relative priority of the FIFOC, SPDIF, AXE, USB ULPI, USB UTMI, SATA internal interrupt signals can be modified.
- The relative priority of the PCI, PCI DMA, PCI MU, FEC, PATA, NFC, LPC, and SDHC internal interrupt signals can be modified.
- The relative priority of the I2C1, I2C2, I2C3, MSCAN1, MSCAN2, BDLC, GPT0, and GPT1 internal interrupt signals can be modified.
- The relative priority of the *ipp_ind_ext_int*[0], *ipp_ind_ext_int*[1] external interrupts, and *ipi_int_internal*[32], *ipi_int_internal*[33], *ipi_int_internal*[34]¹, and *ipi_int_internal*[35] internal interrupts can be modified.
- The relative priority of the PSC0, PSC1, PSC2 and PSC3 internal interrupts can be modified.
- One interrupt source can be assigned to be the programmable highest priority.

All other interrupt sources have a fixed interrupt priority. For details see Table 20-27.

The SIV EC is updated with a 7-bit vector corresponding to the sub-block with the highest current priority.

1. Not available in MPC5123

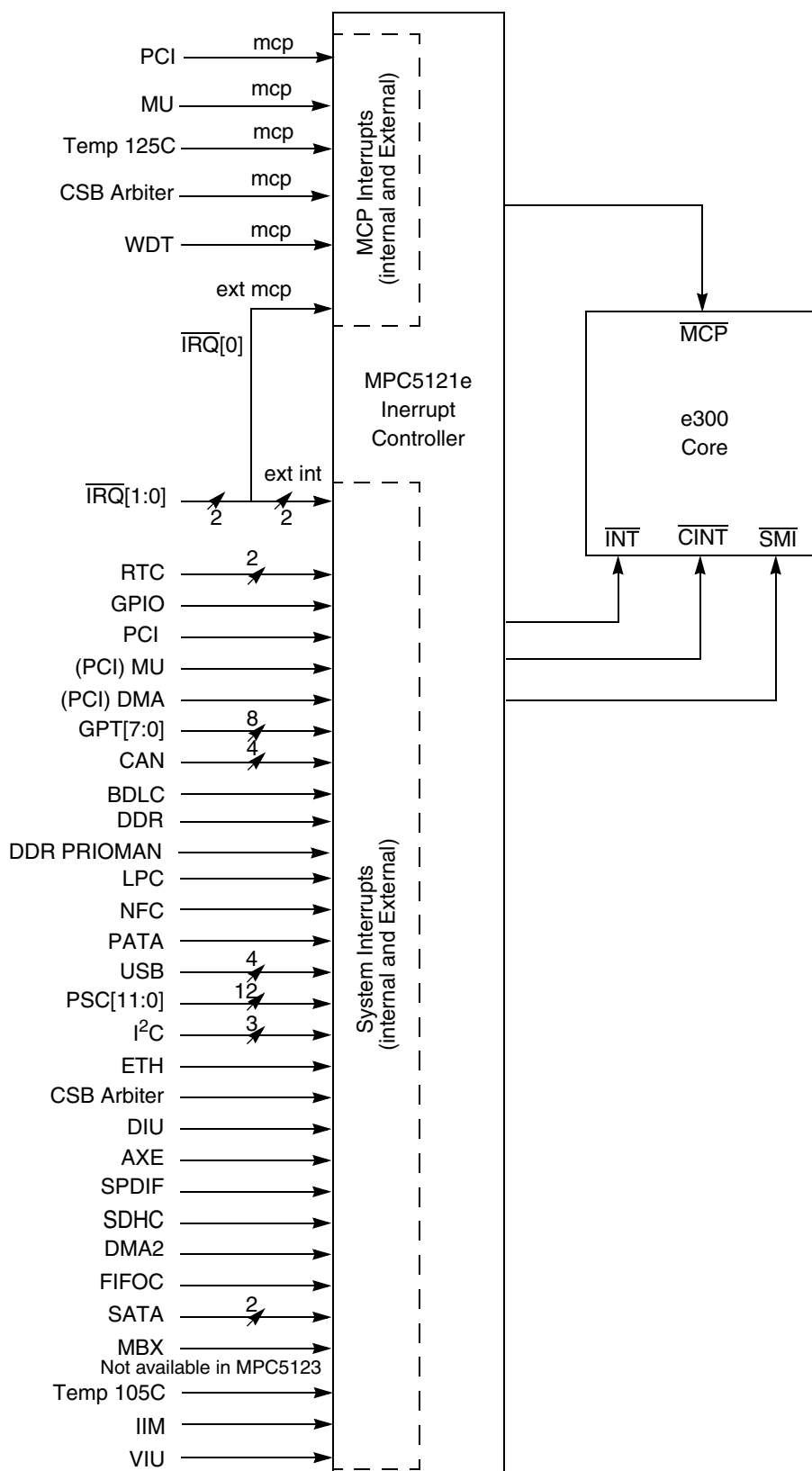


Figure 20-26. MPC5121e Interrupt Structure

20.3.3 Internal Interrupts Group Relative Priority

The relative priority in each internal group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC internal interrupt priority registers (SIPRRx) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the interrupt entries has the following two options:

- **Grouped.** In the group scheme, all interrupts are grouped together at the top of [Table 20-27](#), ahead of most other interrupt sources. This scheme is ideal for applications where all interrupt sources function at a high data rate and interrupt latency is important.
- **Spread.** In the spread scheme, priorities are spread over [Table 20-27](#) so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

20.3.4 Mixed Interrupts Group Relative Priority

The relative priority between up to four internal and four external interrupts in each group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC mixed interrupt priority registers (SMPRRx) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the mixed interrupt entries has the following two options:

- **Grouped.** In the group scheme, all interrupts are grouped together at the top of the priority table, ahead of most other interrupt sources. See [Table 20-27](#) for more information. This scheme is ideal for applications where all interrupt sources function at a high data rate and interrupt latency is important.
- **Spread.** In the spread scheme, priorities are spread over the table so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

20.3.5 Highest Priority Interrupt

In addition to the group relative priority option, SICFR[HPI] can be used to specify one interrupt source as having the highest priority. This interrupt remains within the same interrupt level as the other interrupt controller interrupts, but is serviced before any other interrupt in [Table 20-27](#).

If the highest priority feature is not used, the IPIC selects the interrupt request in MIXA0 to be the highest priority interrupt and the standard interrupt priority order is used from [Table 20-27](#). SICFR[HPI] can be updated dynamically to allow you to change a normally low-priority source into a high-priority source for a period as needed.

20.3.6 Interrupt Source Priorities

Each of the IPIC's internal and external interrupt sources can independently assert one interrupt request to the core. [Table 20-27](#) shows the prioritization of these interrupt sources. As described in previous sections, flexibility exists in the relative ordering of the interrupts, but in general, relative priorities are as shown. A single interrupt priority number is associated with each table entry.

Table 20-27. Interrupt Source Priority Levels (Sheet 1 of 4)

Priority Level	Interrupt Source Description	Multiple Events
0	Highest	—
1	MIXA0 (Spread)	Yes (No for ext. interrupts)
2	MIXA0 (Grouped)	Yes (No for ext. interrupts)
3	MIXA1 (Grouped)	Yes (No for ext. interrupts)
4	MIXA2 (Grouped)	Yes (No for ext. interrupts)
5	MIXA3 (Grouped)	Yes (No for ext. interrupts)
6	MIXB0 (Spread)	Yes (No for ext. interrupts)
7	SYSB0 (Grouped)	Yes
8	SYSB1 (Grouped)	Yes
9	SYSB2 (Grouped)	Yes
10	SYSB3 (Grouped)	Yes
11	MIXA1 (Spread)	Yes (No for ext. interrupts)
12	SYSB4 (Grouped)	Yes
13	SYSB5 (Grouped)	Yes
14	SYSB6 (Grouped)	Yes
15	SYSB7 (Grouped)	Yes
16	MIXB0 (Grouped)	Yes (No for ext. interrupts)
17	MIXB1 (Grouped)	Yes (No for ext. interrupts)
18	MIXB2 (Grouped)	Yes (No for ext. interrupts)
19	MIXB3 (Grouped)	Yes (No for ext. interrupts)
20	MIXB1 (Spread)	Yes (No for ext. interrupts)
21	SYSA0 (Grouped)	Yes
22	SYSA1 (Grouped)	Yes
23	SYSA2 (Grouped)	Yes
24	SYSA3 (Grouped)	Yes
25	MIXA2 (Spread)	Yes (No for ext. interrupts)
26	SYSA4 (Grouped)	Yes
27	SYSA5 (Grouped)	Yes
28	SYSA6 (Grouped)	Yes
29	SYSA7 (Grouped)	Yes
30	MIXA4 (Grouped)	Yes (No for ext. interrupts)
31	MIXA5 (Grouped)	Yes (No for ext. interrupts)
32	MIXA6 (Grouped)	Yes (No for ext. interrupts)
33	MIXA7 (Grouped)	Yes (No for ext. interrupts)

Table 20-27. Interrupt Source Priority Levels (Sheet 2 of 4)

Priority Level	Interrupt Source Description	Multiple Events
34	MIXB2 (Spread)	Yes (No for ext. interrupts)
35	SYSC0 (Grouped)	Yes
36	SYSC1 (Grouped)	Yes
37	SYSC2 (Grouped)	Yes
38	SYSC3 (Grouped)	Yes
39	MIXA3 (Spread)	Yes (No for ext. interrupts)
40	SYSC4 (Grouped)	Yes
41	SYSC5 (Grouped)	Yes
42	SYSC6 (Grouped)	Yes
43	SYSC7 (Grouped)	Yes
44	MIXB4 (Grouped)	Yes (No for ext. interrupts)
45	MIXB5 (Grouped)	Yes (No for ext. interrupts)
46	MIXB6 (Grouped)	Yes (No for ext. interrupts)
47	MIXB7 (Grouped)	Yes (No for ext. interrupts)
48	MIXB3 (Spread)	Yes (No for ext. interrupts)
49	SYSD0 (Grouped)	Yes
50	SYSD1 (Grouped)	Yes
51	SYSD2 (Grouped)	Yes
52	SYSD3 (Grouped)	Yes
53	MIXA4 (Spread)	Yes (No for ext. interrupts)
54	SYSD4 (Grouped)	Yes
55	SYSD5 (Grouped)	Yes
56	SYSD6 (Grouped)	Yes
57	SYSD7 (Grouped)	Yes
58	MIXB4 (Spread)	Yes (No for ext. interrupts)
59	GPT2	Yes
60	SYSB0 (Spread)	Yes
61	SYSA0 (Spread)	Yes
62	GPT3	Yes
63	SYSC0 (Spread)	Yes
64	SYSD0 (Spread)	Yes
65	Reserved	No
66	GPT4	Yes
67	MIXA5 (Spread)	Yes (No for ext. interrupts)

Table 20-27. Interrupt Source Priority Levels (Sheet 3 of 4)

Priority Level	Interrupt Source Description	Multiple Events
68	GPT5	Yes
69	SYSB1 (Spread)	Yes
70	SYSA1 (Spread)	Yes
71	GPT6	Yes
72	SYSC1 (Spread)	Yes
73	SYSD1 (Spread)	Yes
74	Reserved	No
75	GPT7	Yes
76	MIXB5 (Spread)	Yes (No for ext. interrupts)
77	GPIO	Yes
78	SYSB2 (Spread)	Yes
79	SYSA2 (Spread)	Yes
80	RTC SEC	Yes
81	SYSC2 (Spread)	Yes
82	SYSD2 (Spread)	Yes
83	Reserved	No
84	RTC ALARM	Yes
85	MIXA6 (Spread)	Yes (No for ext. interrupts)
86	DDR	Yes
87	SYSB3 (Spread)	Yes
88	SYSA3 (Spread)	Yes
89	SBA	Yes
90	SYSC3 (Spread)	Yes
91	SYSD3 (Spread)	Yes
92	Reserved	No
93	PMC	Yes
94	MIXB6 (Spread)	Yes (No for ext. interrupts)
95	USB ULPI WKUP	Yes
96	SYSB4 (Spread)	Yes
97	SYSA4 (Spread)	Yes
98	USB UTMI WKUP	Yes
99	SYSC4 (Spread)	Yes
100	SYSD4 (Spread)	Yes
101	Reserved	No
102	SATA CMD	Yes

Table 20-27. Interrupt Source Priority Levels (Sheet 4 of 4)

Priority Level	Interrupt Source Description	Multiple Events
103	MIXA7 (Spread)	Yes (No for ext. interrupts)
104	TEMP 105C	Yes
105	SYSB5 (Spread)	Yes
106	SYSA5 (Spread)	Yes
107	IIM	Yes
108	SYSC5 (Spread)	Yes
109	SYSD5 (Spread)	Yes
110	Reserved	No
111	DDR PRIOMAN	Yes
112	MIXB7 (Spread)	Yes (No for ext. interrupts)
113	MSCAN3	Yes
114	SYSB6 (Spread)	Yes
115	SYSA6 (Spread)	Yes
116	MSCAN4	Yes
117	SYSC6 (Spread)	Yes
118	SYSD6 (Spread)	Yes
119	Reserved	No
120	Reserved	Yes
121	Reserved	Yes
122	SYSB7 (Spread)	Yes
123	SYSA7 (Spread)	Yes
124	Reserved	Yes
125	SYSC7 (Spread)	Yes
126	SYSD7 (Spread)	Yes
127	Reserved	No
128	Reserved	Yes

20.3.7 Masking Interrupt Sources

By programming the system interrupt mask registers, SIMSRx and SEMSR, you can mask interrupt requests to the core. Each SIMSRx and SEMSR bit corresponds to an interrupt source. To enable an interrupt, set the corresponding SIMSR or SEMSR bit. When a masked interrupt source has a pending interrupt request, the corresponding SIPNRx or SEMSR bit is set, even though the interrupt is not generated to the core. You can mask all interrupt sources to implement a polling interrupt servicing scheme.

When an interrupt source has multiple interrupting events, you can individually mask these events by programming a mask register within that particular block. [Table 20-27](#) shows which interrupt sources have multiple interrupting events. [Figure 20-27](#) shows an example of how the masking occurs, using a DDR as an example.

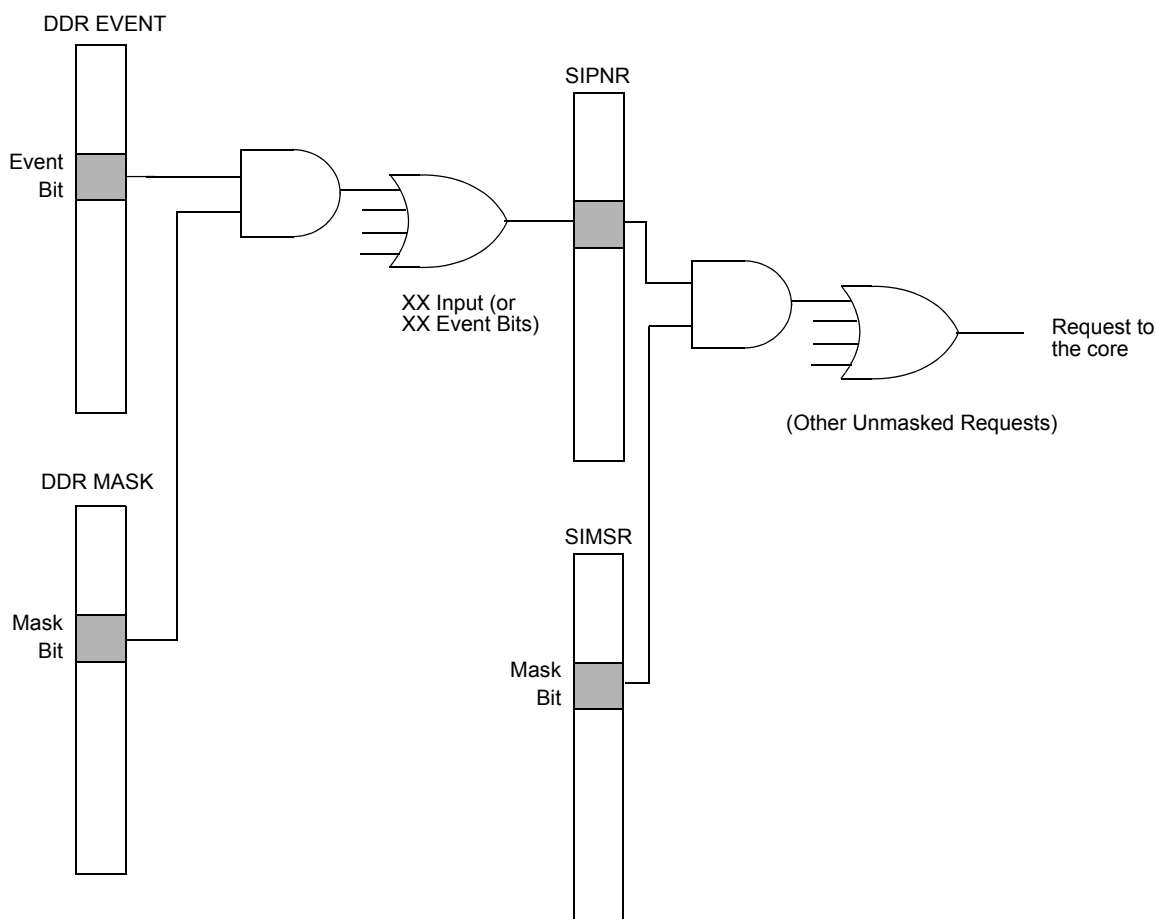


Figure 20-27. DDR Interrupt Request Masking

20.3.8 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core in order of priority according to [Table 20-27](#). The interrupt vector that allows the core to locate the interrupt service routine is made available to the core by interrupt handler software reading SIVCR, SCVCR or SMVCR. The interrupt controller passes an interrupt vector corresponding to the highest-priority, unmasked, pending interrupt in response to a read of SIVCR, SCVCR or SMVCR. [Table 20-4](#) lists the encodings for the seven low-order bits of the interrupt vector.

20.3.9 Machine Check Interrupts

There are 5 non-maskable machine check interrupts (MCP), coming from the internal sources and one programmable MCP from the external source.

Chapter 21

Inter-Integrated Circuit (I²C)

21.1 Introduction

21.1.1 Overview

The Inter-Integrated Circuit (I²C) interface is a two-wire, bidirectional serial bus that provides a simple, efficient method for data exchange among devices. The MPC5121e contains three identical and independent I2C modules.

I²C module operates up to a maximum bus load and timing of 400 Kbps. Also, I²C modules are capable of operating at higher baud rates, up to a maximum frequency equal to $\text{IPS_clock}/20$ with reduced bus loading (where, the value 20 is the minimum SCL Divider in the block).

A maximum bus capacitance of 400 pF limits the maximum communication length and number of possibly connected devices. The module can operate up to a baud rate of 400kbps. This bus is suitable for applications requiring occasional communications over a short distance among a number of devices. It also provides flexibility, allowing more devices to be connected to the bus for further expansion and system development.

I²C is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters attempt to control the bus simultaneously. This feature provides the capability for complex applications with multi-processor control. It may also be used for rapid testing and alignment of end products via external connections to an assembly-line computer. [Figure 21-1](#) shows a block diagram of the I²C module.

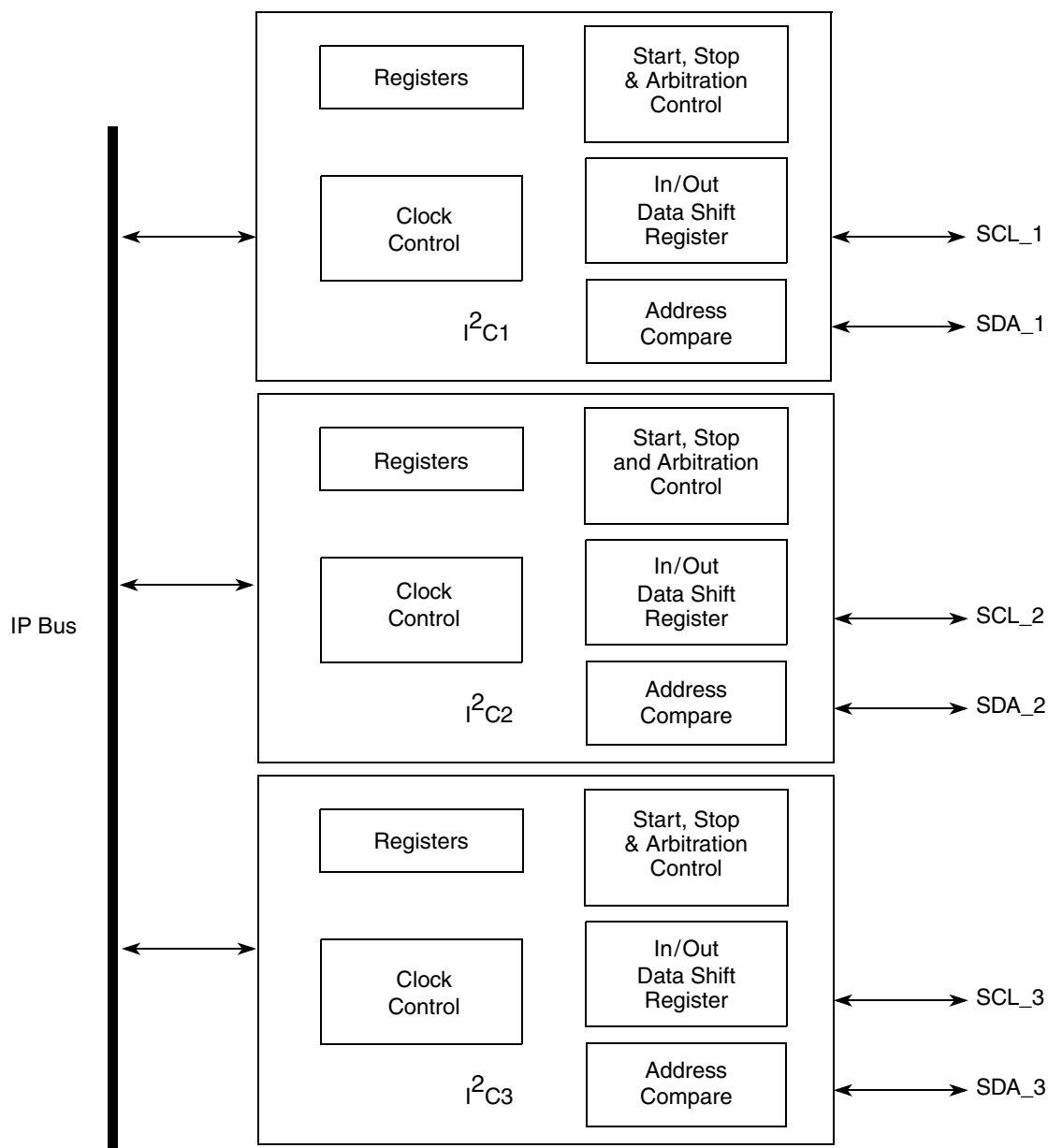


Figure 21-1. Block Diagram – I²C Module

21.1.2 Features

The I²C module has these key features:

- Compatible with I²C bus standard version 2.1
- Multi-master operation
- Software programmable for different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer

- Arbitration loss with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Programmable glitch filter

21.1.3 I²C Controller

The I²C is a simple bidirectional two-wire bus for efficient device-to-device communication. The two wires, serial data line (SDA) and serial clock line (SCL), carry information between this module and other devices connected to the bus. A unique address recognizes each device. Also, each device can operate as transmitter or receiver, depending on the function of the device. In addition to the transmitters and receivers, devices can be masters or slaves. A master is the device that initiates a data transfer on the bus and generates clock signals to permit that transfer. At that time, any device addressed is considered a slave. See [Table 21-1](#).

Table 21-1. I²C Terminology

Term	Description
Transmitter	Device that sends data to bus.
Receiver	Device that receives data from bus.
Master	Device that initiates transfer, generates SCL, and terminates transfer.
Slave	Device that is addressed by master.

Standard communication usually has four functional areas:

- START signal
- Slave address transmission
- Data transfer
- STOP signal

These activities are briefly described in the following sections.

21.1.4 START Signal

A START signal is a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer and wakes up all slaves. Each data transfer may contain several data bytes.

When the bus is free (no master device is engaging the bus), SCL and SDA lines are at a logical high. A master sends a START signal to initiate communications.

21.1.5 STOP Signal

A STOP signal is a low-to-high transition of SDA while SCL is high.

The master generates a STOP signal to terminate communication, which frees the bus. The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

The master can generate a repeated start and address for other devices. At this time, the bus remains busy if a repeated start is generated instead of a stop.

21.1.5.1 Slave Address Transmission

The first byte of data transferred by the master immediately after a START signal is the slave address. This is a 7-bit calling address followed by a R/ \overline{W} bit. The R/ \overline{W} bit tells the slave the desired direction of data transfer.

- 0 = Master writes data (W), becomes transmitter
- 1 = Master reads data (R), becomes receiver

Only a slave with a calling address matching the address transmitted by the master responds by sending back an acknowledge bit. This is done by pulling SDA low at the ninth clock as shown in [Figure 21-2](#).

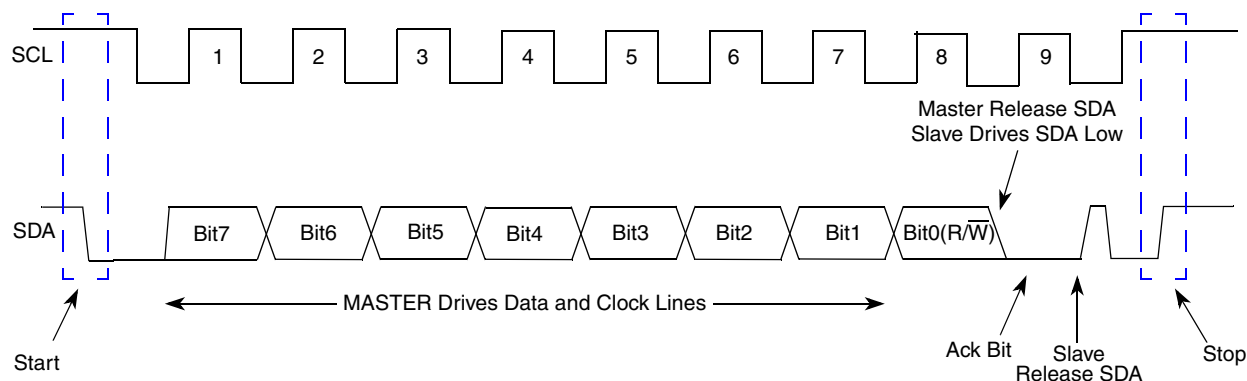


Figure 21-2. Timing Diagram – Start, Address Transfer and Stop Signal

21.1.5.2 Data Transfer

Data transfer proceeds byte-by-byte in a direction specified by the R/ \overline{W} bit sent by the calling master. Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high.

There is one clock pulse on SCL for each data bit. The MSB is transferred first. Each data byte must be followed by an acknowledge bit signalled from the receiving device by pulling SDA low at the ninth clock. One complete 8-bit data byte transfer needs nine clock pulses as shown in [Figure 21-3](#).

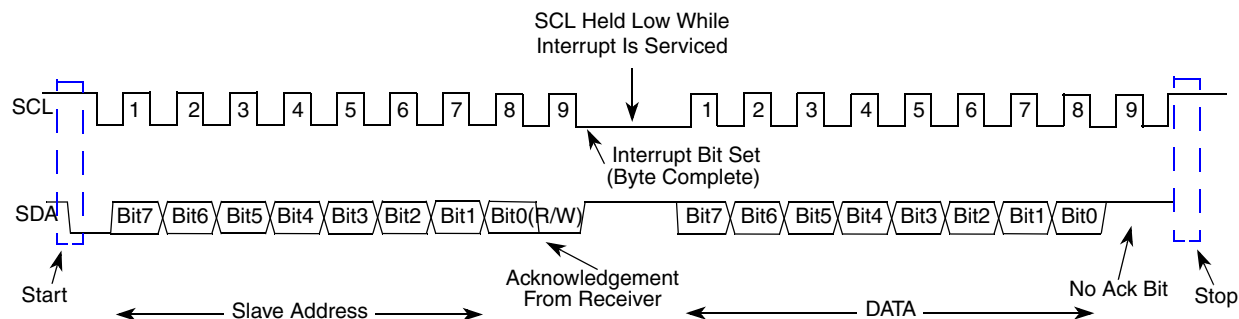


Figure 21-3. Timing Diagram – Start, Address Transfer and Stop Signal

21.1.6 Acknowledge

Figure 21-4 shows the transmitter releasing the SDA line after bit 0 is transmitted. All devices driving the SDA line must have an open-collector configuration. A pull-up register is required to ensure the SDA line goes to a logic 1 when not driven by the master device. The receiver pulls the SDA line low during the acknowledge clock pulse to signal correct reception of the data.

If a slave-receiver does not acknowledge the byte transfer, SDA must be left HIGH by the slave. The master then generates a STOP condition to abort the transfer.

If a master-receiver does not acknowledge the slave transmitter after a byte transmission, it means End-Of-Data (EOD) to the slave. The slave then releases the SDA line for the master to generate a STOP or START signal.

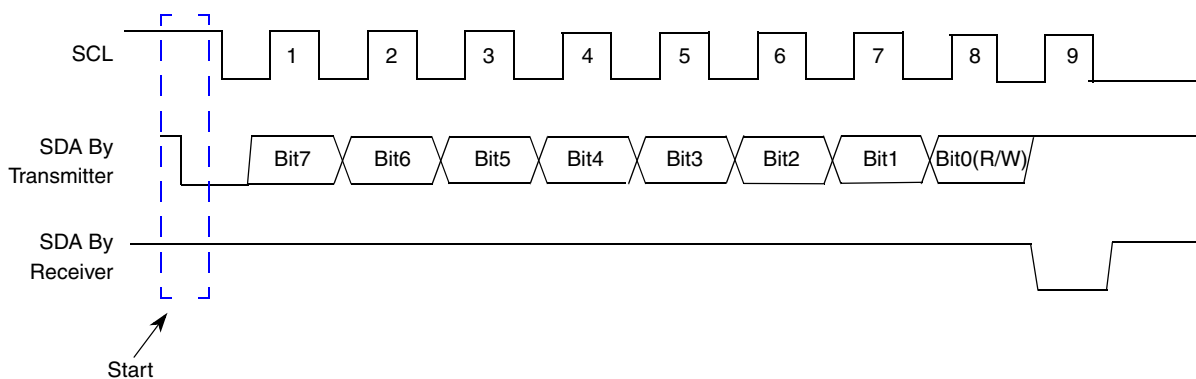


Figure 21-4. Timing Diagram – Receiver Acknowledgement

21.1.6.1 Repeated Start

A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. The master uses this to communicate with another slave or with the same slave in a different mode without releasing the bus.

Various combinations of read/write formats are possible. Figure 21-5 shows examples of:

- The master-transmitter transmitting to a slave-receiver. The transfer direction is not changed.

- The master reading a slave immediately after first byte. At the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes a slave-transmitter.
- The START condition and slave address both repeated using the repeated START signal. This communicates with the same slave in a different mode without releasing the bus. The master transmits data to the slave first, and then the master reads data from the slave by reversing the R/ \overline{W} bit.

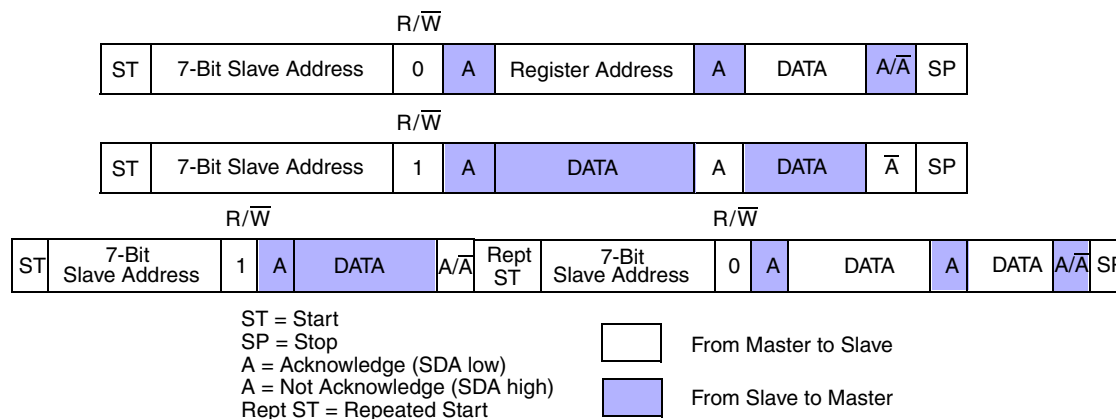


Figure 21-5. Data Transfer, Combined Format

21.1.6.2 Clock Synchronization

I²C is a true multi-master bus. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock.

Because wire-AND logic is used on the SCL line, a high-to-low transition on the SCL line affects all devices connected on the bus. The devices start counting their low period. After a device clock goes low, it holds the SCL line low until the clock high state is reached. However, the change of low-to-high in this device clock may not change the SCL line state if another device clock remains within its low period. Therefore, the synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time. See [Figure 21-6](#).

When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. No difference exists between device clocks and the SCL line state. All devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

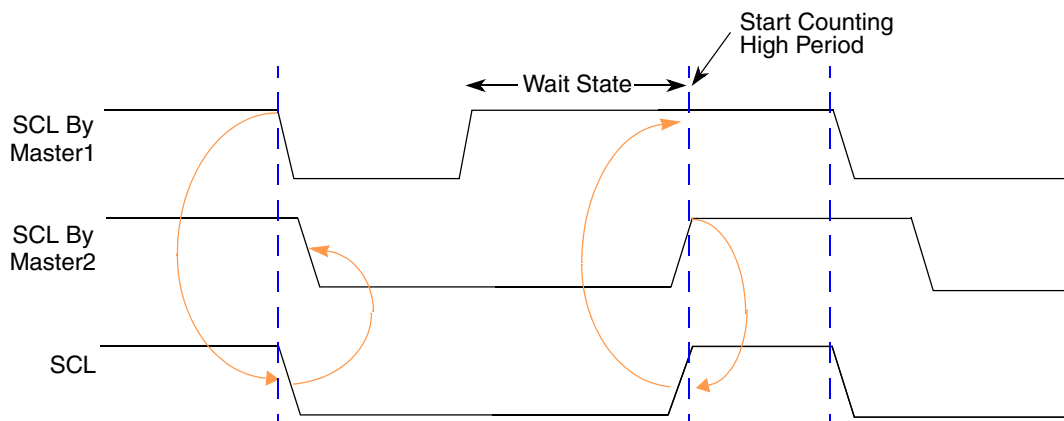


Figure 21-6. Timing Diagram – Clock Synchronization

21.1.7 Arbitration

A data arbitration procedure determines the relative priority of contending masters. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. Losing masters immediately switch to slave-receive mode and stop driving SDA output. In this case, transition from master to slave mode does not generate a STOP condition. A status bit is hardware set to indicate loss of arbitration. Figure 21-7 shows an example of two masters arbitrating for the I²C bus.

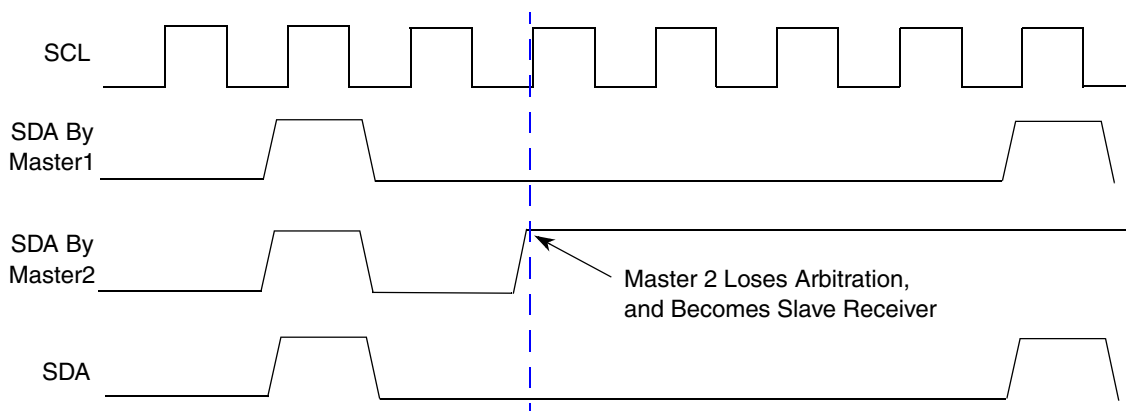


Figure 21-7. Timing Diagram – Arbitration Procedure

21.2 External Signal Description

Table 21-2. Signal Properties

Name	Function	I/O	Reset	Pull Up
SCL	I ² C serial clock line, bidirectional	I/O	1	Yes
SDA	I ² C serial data line, bidirectional	I/O	1	Yes

21.3 Memory Map and Register Definition

The I²C is controlled by 17 32-bit registers. The registers are located at an offset from I²C base address, I²C_BASE, which is memory mapped on the IPBus address. The I²C1 base address (I²C1_base_address) is I²C_BASE + 0x00. The I²C2 base address (I²C2_base_address) is I²C_BASE + 0x20. The I²C3 base address (I²C3_base_address) is I²C_BASE + 0x40. Each of their own private registers addresses are relative to their base offset. There are two common registers for the three I²C modules: I²C interrupt control register and glitch filter control register.

The I²C Interface registers are provided below:

Table 21-3. I²C Block Memory Map

Offset	Register	Access	Section/Page
General Registers			
0x00	MADR1 – I ² C1 Address Register	R/W	21.3.1.1/21-9
0x04	MFDR1 – I ² C1 Frequency Divider Register	R/W	21.3.1.2/21-10
0x08	MCR1 – I ² C1 Control Register	R/W	21.3.1.3/21-21
0x0C	MSR1 – I ² C1 Status Register	R/W	21.3.1.4/21-23
0x10	MDR1 – I ² C1 Date I/O Register	R/W	21.3.1.5/21-26
0x20	MADR2 – I ² C2 Address Register	R/W	21.3.1.1/21-9
0x24	MFDR2 – I ² C2 Frequency Divider Register	R/W	21.3.1.2/21-10
0x28	MCR2 – I ² C2 Control Register	R/W	21.3.1.3/21-21
0x2C	MSR2 – I ² C2 Status Register	R/W	21.3.1.4/21-23
0x30	MDR2 – I ² C2 Date I/O Register	R/W	21.3.1.5/21-26
0x40	MADR3 – I ² C3 Address Register	R/W	21.3.1.1/21-9
0x44	MFDR3 – I ² C3 Frequency Divider Register	R/W	21.3.1.2/21-10
0x48	MCR3 – I ² C3 Control Register	R/W	21.3.1.3/21-21
0x4C	MSR3 – I ² C3 Status Register	R/W	21.3.1.4/21-23
0x50	MDR3 – I ² C3 Date I/O Register	R/W	21.3.1.5/21-26
0x60	MSB – I ² C Interrupt Control Register	R/W	21.3.1.6/21-27
0x64	MIFR – I ² C Filter Register	R/W	21.3.1.7/21-28

21.3.1 Register Descriptions

21.3.1.1 I²C Address Register (MADR)

Offset I²C1/2/3_base_address + 0x00/0x20/0x40

Access:

User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ADR[7:1]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 21-8. I²C Address Register (MADR)

Table 21-4. MADR Field Descriptions

Field	Description
ADR[7:1]	Bits 0 to 6 contains the address I ² C responds to when addressed as a slave. This is not the address sent on the bus during address transfer.

21.3.1.2 I²C Frequency Divider Register (MFDR)

Offset I²C1/2/3_base_address + 0x04/0x24/0x44

Access: User read/write

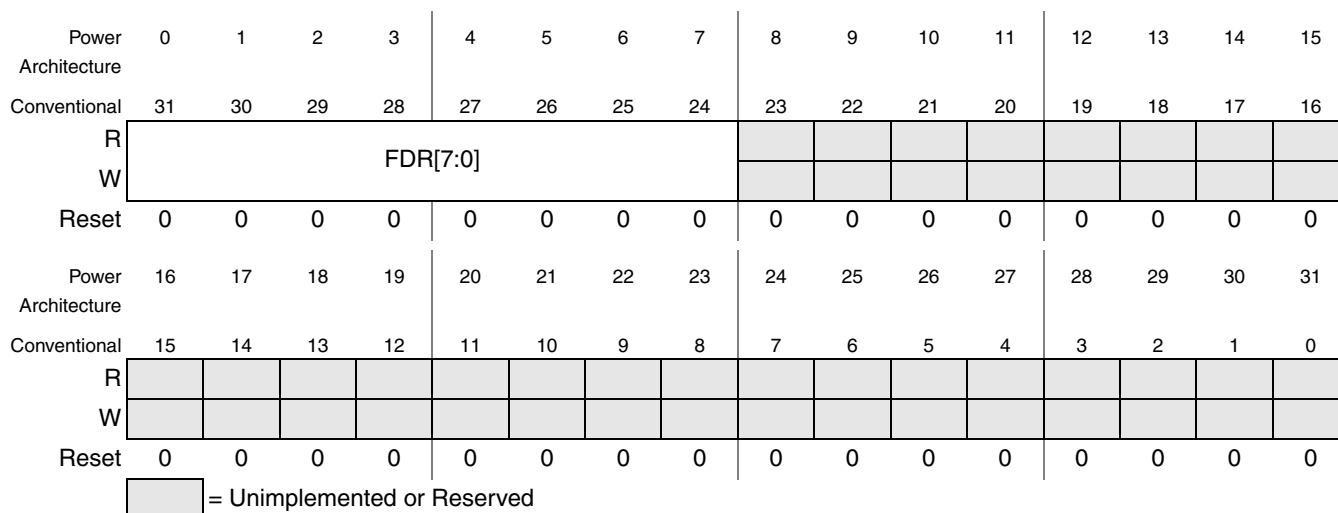
Figure 21-9. I²C Frequency Divider Register (MFDR)

Table 21-5. MFDR Field Descriptions

Field	Description
FDR[7:6]	These two bits act as a prescale divider of the input module clock.
FDR[5:0]	This field prescales the clock for bit-rate selection.

The frequency divide register determines the SCL or serial bit-clock frequency. [Table 21-5](#) must be used to select FDR bits that produce an appropriate SCL. The following relationships, one through four, illustrate the connection between [Table 21-6](#) and the signals in the I²C timing specification.

1. $SCL \text{ (in kHz)} = (1/1000) * [IPS \text{ clock speed (in Hz)}] / (SCL \text{ Period})$
2. $SDA \text{ Hold Time (in us)} = 1000 * (SDA \text{ Hold} / SCL \text{ Period}) / [SCL \text{ (in kHz)}]$
3. $SCL \text{ Hold Time of START (in us)} = 1000 * (SDA \text{ Hold of START} / SCL \text{ Period}) / [SCL \text{ (in kHz)}]$
4. $SCL \text{ Hold Time of STOP (in us)} = 1000 * (SDA \text{ Hold of STOP} / SCL \text{ Period}) / [SCL \text{ (in kHz)}]$

[Figure 21-10](#) illustrates the relationship between IPS clock and the I²C signals.

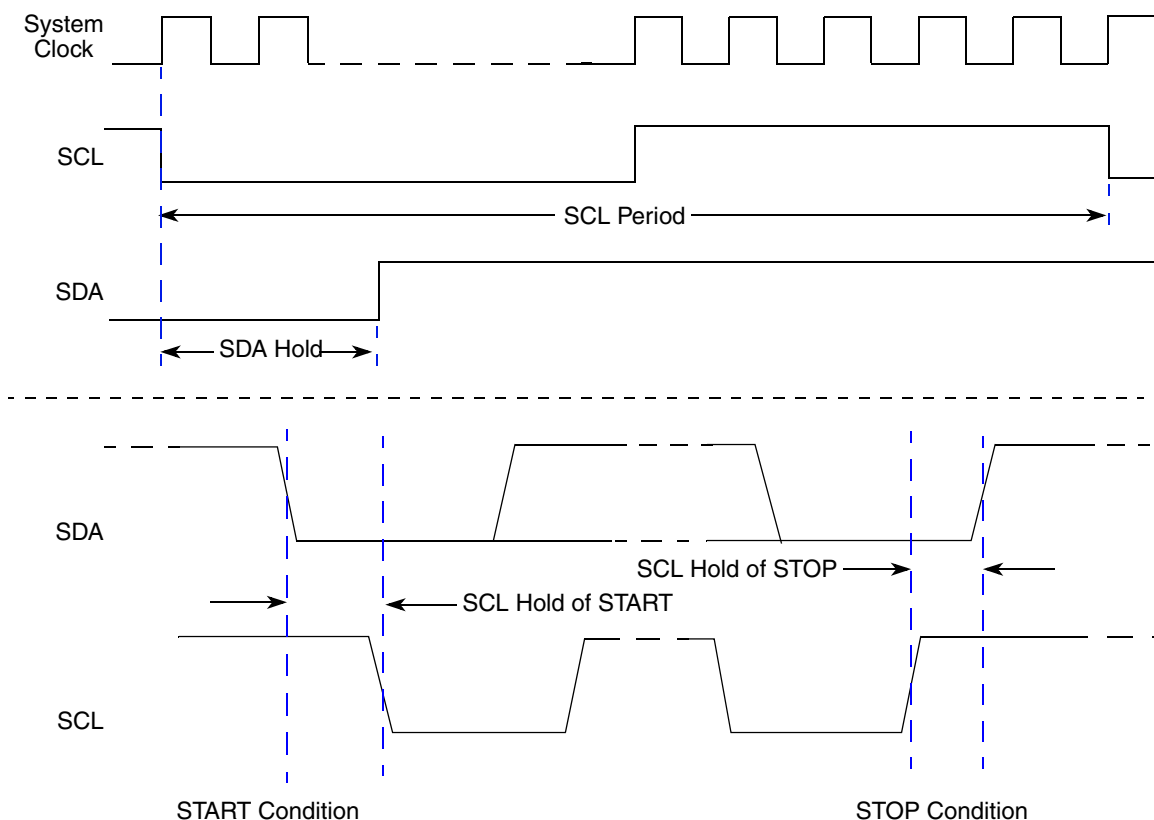


Figure 21-10. Timing Diagram of I²C Signal Relationships

For standard mode I²C, the I²C specification states:

$$(SCL \leq 100 \text{ kHz}) \quad \text{Eqn. 21-1}$$

and

$$(0.3 \mu\text{s} \leq \text{SDA Hold Time} \leq 3.45 \mu\text{s}) \quad \text{Eqn. 21-2}$$

and

$$(\text{SCL Hold of START} \geq 4 \mu\text{s}) \quad \text{Eqn. 21-3}$$

and

$$(\text{SCL Hold of STOP} \geq 4 \mu\text{s}) \quad \text{Eqn. 21-4}$$

This means the system programmer must choose SCL Period, SDA Hold, SCL Hold of START, and SCL Hold of STOP from [Table 21-6](#) to satisfy [Equation 21-5](#) through [Equation 21-8](#).

$$\text{SCL Period} \geq (1/100,000) * [\text{IPS clock speed (in Hz)}] \quad \text{Eqn. 21-5}$$

and

$$(0.0003) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \leq \text{SDA Hold} \leq (0.00345) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-6}$$

and

$$\text{SCL Hold of START} \geq (0.004) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-7}$$

and

$$\text{SCL Hold of STOP} \geq (0.004) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-8}$$

In this case, the simplest strategy for the system programmer to use is:

1. Identify all rows of [Table 21-6](#) where SCL period satisfies criteria in [Equation 21-5](#). This set of rows limits the choices of SCL allowed for this particular IPS clock.
2. Calculate the SCL associated with these rows according to [Equation 21-1](#) and decide which speeds are acceptable (fast enough or slow enough) for the system.
3. Find the subset of those rows associated with the acceptable I²C clock speeds such that SDA hold satisfies criteria in [Equation 21-6](#).
4. Choose the preferred FDR setting from among the subset that meets [Equation 21-5](#) and [Equation 21-6](#).
5. Check that the preferred FDR setting also satisfies [Equation 21-7](#) and [Equation 21-8](#). Usually, it does. If not, choose a different FDR setting that meets [Equation 21-5](#), [Equation 21-6](#), [Equation 21-7](#), and [Equation 21-8](#).

Likewise, for fast mode I²C, it must also meet the fast-mode I²C bus specification.

$$(\text{SCL} \leq 400 \text{ kHz}) \quad \text{Eqn. 21-9}$$

and

$$(0.3 \text{ } \mu\text{s} \leq \text{SDA Hold Time} \leq 0.9 \text{ } \mu\text{s}) \quad \text{Eqn. 21-10}$$

and

$$(\text{SCL Hold of START} \geq 0.6 \text{ } \mu\text{s}) \quad \text{Eqn. 21-11}$$

and

$$(\text{SCL Hold of STOP} \geq 0.6 \text{ } \mu\text{s}) \quad \text{Eqn. 21-12}$$

That means the system programmer must choose SCL Period, SDA hold, SCL hold of START, and SCL hold of STOP from [Table 21-6](#) to satisfy [Equation 21-9](#) through [Equation 21-12](#):

$$\text{SCL Period} \geq (1/400,000) * [\text{IPS clock speed (in Hz)}] \quad \text{Eqn. 21-13}$$

and

$$(0.0003) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \leq \text{SDA Hold} \leq (0.0009) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-14}$$

and

$$\text{SCL Hold of START} \geq (0.0006) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-15}$$

and

$$\text{SCL Hold of STOP} \geq (0.0006) * [\text{SCL (in kHz)}] * (\text{SCL Period}) \quad \text{Eqn. 21-16}$$

In this case, the strategy to choose FDR value is like the one for standard mode device, but based on the different timing requirements.

In [Table 21-6](#), the SCL Divider has the same meaning as SCL Period shown in [Figure 21-10](#), which uses the IPS clock as a time unit. For example, from the table, if the SCL Divider equals 20, the SCL Period is equal to 20 IPS clocks.

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 1 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0x20	20	7	6	11
0x21	22	7	7	12
0x22	24	8	8	13
0x23	26	8	9	14
0x24	28	7	10	15
0x00	28	9	10	15
0x01	30	9	11	16
0x25	32	7	12	17
0x02	34	10	13	18
0x26	36	9	14	19
0x27	40	9	16	21
0x03	40	10	16	21
0x60	40	14	12	22
0x04	44	11	18	23
0x61	44	14	14	24
0x28	48	9	18	25
0x05	48	11	20	25
0x62	48	16	16	26
0x63	52	16	18	28
0x29	56	9	22	29
0x06	56	13	24	29
0x64	56	14	20	30
0x40	56	18	20	30
0x41	60	18	22	32
0x2A	64	13	26	33
0x65	64	14	24	34
0x07	68	13	30	35

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 2 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0x42	68	20	26	36
0x2B	72	13	30	37
0x66	72	18	28	38
0x2C	80	9	38	41
0x08	80	17	34	41
0x67	80	18	32	42
0x43	80	20	32	42
0xA0	80	28	24	44
0xE0	80	28	24	44
0x09	88	17	38	45
0x44	88	22	36	46
0xA1	88	28	28	48
0xE1	88	28	28	48
0x2D	96	9	46	49
0x68	96	18	36	50
0x45	96	22	40	50
0xA2	96	32	32	52
0xE2	96	32	32	52
0x0A	104	21	46	53
0xA3	104	32	36	56
0xE3	104	32	36	56
0x2E	112	17	54	57
0x69	112	18	44	58
0x46	112	26	48	58
0xA4	112	28	40	60
0xE4	112	28	40	60
0x80	112	36	40	60
0xC0	112	36	40	60
0x81	120	36	44	64
0xC1	120	36	44	64
0x2F	128	17	62	65
0x0B	128	21	58	65

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 3 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0x6A	128	26	52	66
0xA5	128	28	48	68
0xE5	128	28	48	68
0x47	136	26	60	70
0x82	136	40	52	72
0xC2	136	40	52	72
0x0C	144	25	70	73
0x6B	144	26	60	74
0xA6	144	36	56	76
0xE6	144	36	56	76
0x30	160	17	78	81
0x6C	160	18	76	82
0x0D	160	25	78	81
0x48	160	34	68	82
0xA7	160	36	64	84
0xE7	160	36	64	84
0x83	160	40	64	84
0xC3	160	40	64	84
0x49	176	34	76	90
0x84	176	44	72	92
0xC4	176	44	72	92
0x31	192	17	94	97
0x6D	192	18	92	98
0x0E	192	33	94	97
0xA8	192	36	72	100
0xE8	192	36	72	100
0x85	192	44	80	100
0xC5	192	44	80	100
0x4A	208	42	92	106
0x32	224	33	110	113
0x6E	224	34	108	114
0xA9	224	36	88	116

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 4 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xE9	224	36	88	116
0x86	224	52	96	116
0xC6	224	52	96	116
0x0F	240	33	118	121
0x33	256	33	126	129
0x6F	256	34	124	130
0x4B	256	42	116	130
0xAA	256	52	104	132
0xEA	256	52	104	132
0x87	272	52	120	140
0xC7	272	52	120	140
0x10	288	49	142	145
0x4C	288	50	140	146
0xAB	288	52	120	148
0xEB	288	52	120	148
0x34	320	33	158	161
0x70	320	34	156	162
0xAC	320	36	152	164
0xEC	320	36	152	164
0x11	320	49	158	161
0x4D	320	50	156	162
0x88	320	68	136	164
0xC8	320	68	136	164
0x89	352	68	152	180
0xC9	352	68	152	180
0x35	384	33	190	193
0x71	384	34	188	194
0xAD	384	36	184	196
0xED	384	36	184	196
0x12	384	65	190	193
0x4E	384	66	188	194
0x8A	416	84	184	212

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 5 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xCA	416	84	184	212
0x36	448	65	222	225
0x72	448	66	220	226
0xAE	448	68	216	228
0xEE	448	68	216	228
0x13	480	65	238	241
0x4F	480	66	236	242
0x37	512	65	254	257
0x73	512	66	252	258
0xAF	512	68	248	260
0xEF	512	68	248	260
0x8B	512	84	232	260
0xCB	512	84	232	260
0x14	576	97	286	289
0x50	576	98	284	290
0x8C	576	100	280	292
0xCC	576	100	280	292
0x38	640	65	318	321
0x74	640	66	316	322
0xB0	640	68	312	324
0xF0	640	68	312	324
0x15	640	97	318	321
0x51	640	98	316	322
0x8D	640	100	312	324
0xCD	640	100	312	324
0x39	768	65	382	385
0x75	768	66	380	386
0xB1	768	68	376	388
0xF1	768	68	376	388
0x16	768	129	382	385
0x52	768	130	380	386
0x8E	768	132	376	388

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 6 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xCE	768	132	376	388
0x3A	896	129	446	449
0x76	896	130	444	450
0xB2	896	132	440	452
0xF2	896	132	440	452
0x17	960	129	478	481
0x53	960	130	476	482
0x8F	960	132	472	484
0xCF	960	132	472	484
0x3B	1024	129	510	513
0x77	1024	130	508	514
0xB3	1024	132	504	516
0xF3	1024	132	504	516
0x18	1152	193	574	577
0x54	1152	194	572	578
0x90	1152	196	568	580
0xD0	1152	196	568	580
0x3C	1280	129	638	641
0x78	1280	130	636	642
0xB4	1280	132	632	644
0xF4	1280	132	632	644
0x19	1280	193	638	641
0x55	1280	194	636	642
0x91	1280	196	632	644
0xD1	1280	196	632	644
0x3D	1536	129	766	769
0x79	1536	130	764	770
0xB5	1536	132	760	772
0xF5	1536	132	760	772
0x1A	1536	257	766	769
0x56	1536	258	764	770
0x92	1536	260	760	772

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 7 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xD2	1536	260	760	772
0x3E	1792	257	894	897
0x7A	1792	258	892	898
0xB6	1792	260	888	900
0xF6	1792	260	888	900
0x1B	1920	257	958	961
0x57	1920	258	956	962
0x93	1920	260	952	964
0xD3	1920	260	952	964
0x3F	2048	257	1022	1025
0x7B	2048	258	1020	1026
0xB7	2048	260	1016	1028
0xF7	2048	260	1016	1028
0x1C	2304	385	1150	1153
0x58	2304	386	1148	1154
0x94	2304	388	1144	1156
0xD4	2304	388	1144	1156
0x7C	2560	258	1276	1282
0xB8	2560	260	1272	1284
0xF8	2560	260	1272	1284
0x1D	2560	385	1278	1281
0x59	2560	386	1276	1282
0x95	2560	388	1272	1284
0xD5	2560	388	1272	1284
0x7D	3072	258	1532	1538
0xB9	3072	260	1528	1540
0xF9	3072	260	1528	1540
0x1E	3072	513	1534	1537
0x5A	3072	514	1532	1538
0x96	3072	516	1528	1540
0xD6	3072	516	1528	1540
0x7E	3584	514	1788	1794

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 8 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xBA	3584	516	1784	1796
0xFA	3584	516	1784	1796
0x1F	3840	513	1918	1921
0x5B	3840	514	1916	1922
0x97	3840	516	1912	1924
0xD7	3840	516	1912	1924
0x7F	4096	514	2044	2050
0xBB	4096	516	2040	2052
0xFB	4096	516	2040	2052
0x5C	4608	770	2300	2306
0x98	4608	772	2296	2308
0xD8	4608	772	2296	2308
0xBC	5120	516	2552	2564
0xFC	5120	516	2552	2564
0x5D	5120	770	2556	2562
0x99	5120	772	2552	2564
0xD9	5120	772	2552	2564
0xBD	6144	516	3064	3076
0xFD	6144	516	3064	3076
0x5E	6144	1026	3068	3074
0x9A	6144	1028	3064	3076
0xDA	6144	1028	3064	3076
0xBE	7168	1028	3576	3588
0xFE	7168	1028	3576	3588
0x5F	7680	1026	3836	3842
0x9B	7680	1028	3832	3844
0xDB	7680	1028	3832	3844
0xBF	8192	1028	4088	4100
0xFF	8192	1028	4088	4100
0x9C	9216	1540	4600	4612
0xDC	9216	1540	4600	4612
0x9D	10240	1540	5112	5124

Table 21-6. I²C Frequency Divider Bit Selection (Sheet 9 of 9)

FDR Value	SCL Divider	SDA Hold	SCL Hold of START	SCL Hold of STOP
0xDD	10240	1540	5112	5124
0x9E	12288	2052	6136	6148
0xDE	12288	2052	6136	6148
0x9F	15360	2052	7672	7684
0xDF	15360	2052	7672	7684

21.3.1.3 I²C Control Register (MCR)

Offset I²C1/2/3_base_address + 0x08/0x28/0x48

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN	IEN	STA	TX	TXAK	RSTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 21-11. I²C Control Register (MCR)
(Register repeats for reference.)

Table 21-7. MCR Field Descriptions

Field	Description
EN	<p>I²C Enable. Bit controls software reset of entire I²C module. If I²C module is enabled in the middle of a byte transfer, interface behaves as follows:</p> <p>Slave mode ignores current bus transfer and starts operating when a subsequent start condition is detected. Master mode is not aware if bus is busy. If a start cycle is initiated, current bus cycle may become corrupt. Ultimately, this results in the current bus master or I²C module losing arbitration, after which bus operation returns to normal.</p> <p>0 Module is reset and disabled. This is the Power-ON reset. When low the interface is held in reset, but registers can continue to be accessed.</p> <p>1 I²C module is enabled. Bit must be set before other CR bits have any effect.</p>
IEN	<p>I²C Interrupt Enable</p> <p>0 Interrupts from I²C module are disabled. This does not clear currently pending interrupt conditions.</p> <p>1 Interrupts from I²C module are enabled. An I²C interrupt occurs, provided the status register IF bit is also set.</p>

Inter-Integrated Circuit (I2C)

Offset I²C1/2/3_base_address + 0x08/0x28/0x48

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	EN	IEN	STA	TX	TXAK	RSTA										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 21-11. I²C Control Register (MCR)
(Register repeats for reference.)

Table 21-7. MCR Field Descriptions (continued)

Field	Description
STA	<p>Master/Slave Mode Select. Bit clears on reset. When bit changes from 0 to 1, a START signal is generated on the bus and master mode is selected.</p> <p>When bit changes from 1 to 0, a STOP signal is generated and operation mode changes from master to slave. STA is cleared without generating a STOP signal when the master loses arbitration.</p> <p>0 Slave Mode 1 Master Mode</p>
TX	<p>Transmit/Receive Mode Select. Bit selects master/slave transfer direction. When addressed as slave, software should set according to status register SRW bit.</p> <p>When in master mode, bit should be set according to type of transfer required.</p> <p>For address cycles, bit is always high.</p> <p>0 Receive 1 Transmit</p>
TXAK	<p>Transmit Acknowledge Enable. Bit specifies value driven to SDA during acknowledge cycles for master and slave receivers. Values are used only when I²C is a receiver, not a transmitter.</p> <p>0 Acknowledge signal is sent to bus at 9th clock bit after receiving 1 Byte of data. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1)</p>
RSTA	<p>Repeat Start. Writing 1 to this bit generates a repeated START condition on the bus, provided it is the current bus master. Bit is always read low.</p> <p>If the bus is owned by another master, attempting a repeated start at the wrong time results in loss of arbitration.</p> <p>1 Generate repeat start cycle</p>

21.3.1.4 I²C Status Register (MSR)

Offset I²C1/2/3_base_address + 0x0C/0x2C/0x4C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CF	AAS	BB	AL	AKF	SRW	IF	RXAK								
W																
Reset	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 21-12. I²C Status Register (MSR)
(The register is repeated for reference.)

Table 21-8. MSR Field Descriptions¹

Field	Description
CF	Data transferring. Bit clears while 1 byte of data is being transferred. Bit is set by falling edge of ninth clock of a byte transfer. 0 Transfer in progress 1 Transfer complete
AAS	Addressed As Slave. Bit sets when its own specific address (I ² C Address Register) is matched with the calling address. The CPU is interrupted provided IEN is set. The CPU needs to check the SRW bit and set its Tx/Rx mode accordingly. Writing to the I ² C control register clears this bit. 0 Not addressed 1 Addressed as a slave
BB	Bus Busy. Bit indicates bus status. When a START signal is detected, BB is set. If a STOP signal is detected, it is cleared. 0 Bus is idle 1 Bus is busy
AL	Arbitration Lost. Hardware sets the bit when the arbitration procedure is lost. Arbitration is lost in the following circumstances: <ul style="list-style-type: none"> • SDA sampled low when master drives high during an address or data Tx cycle. • SDA sampled low when master drives high during a data Rx cycle acknowledge bit. • Start cycle is attempted when bus is busy. • A repeated start cycle is requested in slave mode. • Stop condition is detected when not requested by master. 0 No arbitration lost 1 Arbitration lost Software must clear this bit by writing 0 in the interrupt routine after detecting arbitration lost and interrupt flag(IF) bit is asserted.

Offset I²C1/2/3_base_address + 0x0C/0x2C/0x4C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CF	AAS	BB	AL	AKF	SRW	IF	RXAK								
W																
Reset	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 21-12. I²C Status Register (MSR)
 (The register is repeated for reference.)

Table 21-8. MSR Field Descriptions¹ (continued)

Field	Description
AKF	<p>Acknowledge Cycle Falling Edge when Arbitration Lost and Addressed as Slave. Hardware sets the bit upon the falling edge of the acknowledge cycle after arbitration has been lost and addressed as slave. In this specific case, the interrupt (IF=1) is really the second one set by the hardware (which is a side-effect of a fix to make the I2C module fully I2C-spec compliant - see note in section 18.5.3 Special note on AKF). The software must use this bit to distinguish if the interrupt is the first one (set upon rising edge of acknowledge cycle) or the second one (set upon falling edge of acknowledge cycle). The software should only take action for AL and AAS if the interrupt is the second one, the traditional time for the interrupt.</p> <p>0 First interrupt on rising edge of acknowledge cycle – software should not take AL and AAS action (see later section for typical software flow diagram).</p> <p>1 Second interrupt on falling edge of acknowledge – software should take AL and AAS action</p> <p>If AKF bit is set to 1 by hardware, Software must clear this bit by writing it 0 in the interrupt routine.</p>
SRW	<p>Slave Read/Write. When set, bit indicates the R/W command bit value of the calling address sent from the master.</p> <p>Note: Bit is valid only when I²C is in slave mode, a complete address transfer occurred with an address match, and no other transfers were initiated. Checking this bit, the CPU can select slave Tx/Rx mode according to the master command.</p> <p>0 Slave receive, master writing to slave</p> <p>1 Slave transmit, master reading from slave</p>
IF	<p>I²C Interrupt. Sets when an interrupt is pending. If IEN is set, a processor interrupt request is generated. IF sets when one of the following events occurs:</p> <ul style="list-style-type: none"> • Complete 1-Byte transfer (set at falling edge of ninth clock). • A Rx calling address matches its own specific address in slave mode. • Arbitration is lost. <p>0 No interrupt is generated</p> <p>1 An interrupt is generated under the above listed condition.</p> <p>Software must clear this interrupt bit by writing it 0 in the interrupt routine.</p>

Offset I²C1/2/3_base_address + 0x0C/0x2C/0x4C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CF	AAS	BB	AL	AKF	SRW	IF	RXAK								
W																
Reset	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 21-12. I²C Status Register (MSR)
 (The register is repeated for reference.)

Table 21-8. MSR Field Descriptions¹ (continued)

Field	Description
RXAK	<p>Receive Acknowledge. SDA value during the bus cycle acknowledge bit.</p> <p>If bit is low, it indicates an acknowledge signal was received after completion of 8 bits of data transmission on the bus.</p> <p>If bit is high, it means no acknowledge signal is detected at the 9th clock.</p> <p>0 Acknowledge received</p> <p>1 No acknowledge received</p>

¹ This status register is read-only with the exception of bit 6 (IF), bit 4 (AKF) and bit 3 (AL), which are software clearable by writing 0. Writing 1 to these three bits (AL, AKF and IF bit), i2c does nothing, but writing 0 to them, i2c does the clear operation. To avoid clearing these three flags unintentionally, force the cleared bit to 0 and the non-cleared bit should be set to 1. For example, if you want to clear the AL bit, only the AL bit must be set to 0 (AL=0). The other two flags should be equal to 1 (AKF=1, IF=1).

21.3.1.5 I²C Data I/O Register (MDR)

Offset I²C1/2/3_base_address + 0x10/0x30/0x50

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	D7	D6	D5	D4	D3	D2	D1	D0								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 21-13. I²C Data I/O Register (MDR)

Table 21-9. MDR Field Descriptions

Field	Description
D[7:0]	<p>Master Transmit Mode. When data is written to this register, a data transfer is initiated. The most significant bit is sent first.</p> <p>Note: In this mode, the first data byte written to DR. Assertion of STA is used for the address transfer and should be comprised of the calling address (in position D[7]:D[1]) concatenated with the required R/\overline{W} bit (in position D0).</p> <p>In Master Receive Mode, reading this register initiates next byte data receiving.</p> <p>In Slave Mode, the same functions are available after an address match occurs.</p>

21.3.1.6 I²C Interrupt Control Register (ICR)

Offset I²C_base_address + 0x60

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R			BNBE3	IE3	BNBE2	IE2	BNBE1	IE1								
W																
Reset	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 21-14. I²C Interrupt Control Register (ICR)

Table 21-10. ICR Field Descriptions

Field	Description
BNBE3	Bus Not Busy Enable 3. This bit lets module 3 generate an interrupt when the bus is not busy. BNBE3 indicates an idle condition. To clear the interrupt, software must write 0 to the bit position. Reset condition disables BNBE3.
IE3	Interrupt Enable 3, This bit routes the interrupt for module 3 to the CPU. Clear by writing 0 to this bit position. Reset condition enables IE3.
BNBE2	Bus Not Busy Enable 2, This bit lets module 2 generate an interrupt when the bus is not busy. BNBE2 indicates an idle condition. To clear the interrupt, software must write 0 to the bit position. Reset condition disables BNBE2.
IE2	Interrupt Enable 2, This bit routes the interrupt for module 2 to the CPU. Clear by writing 0 to this bit position. Reset condition enables IE2.
BNBE1	Bus Not Busy Enable 1, This bit lets module 1 generate an interrupt when the bus is not busy. BNBE1 indicates an idle condition. To clear the interrupt, software must write 0 to the bit position. Reset condition disables this bit.
IE1	Interrupt Enable 1, This bit routes the interrupt for module 1 to the CPU. Clear by writing 0 to this bit position. Reset condition enables IE1.

The interrupt control register is common to three MPC5121e I²C modules. Each module generates an internal interrupt that can be routed to the CPU if the respective IE bit is set to 1.

Reset condition is IE set, and all other enable bits clear.

The BNBE bit lets the module generate an interrupt when the bus becomes not-busy. This implies receipt of a STOP condition, for which the module normally does not generate an interrupt. Because bus-not-busy

is an idle condition, it is necessary for software responding to this interrupt to clear the BNBE bit to clear the interrupt condition. Otherwise, the interrupt condition persists until another I²C transaction is initiated.

21.3.1.7 I²C Filter Register (MIFR)

Offset I ² C_base_address + 0x64												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					FR3	FR2	FR1	FR0								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 21-15. I²C Filter Register (MIFR)

Table 21-11. MIFR Field Descriptions

Field	Description
FR[7:4]	Bits 7 to 4 contain the programming controls for the width of glitch (in terms of IPS clock cycles) the filter should absorb; in other words, the filter does not let glitches less than or equal to this width setting pass. FR[] 3210 0000 No Filter/Bypass 0001 Filter glitches up to width of 1 IPS clock cycle 0010 Filter glitches up to width of 2 IPS clock cycles 0011 Filter glitches up to width of 3 IPS clock cycles 0100 Filter glitches up to width of 4 IPS clock cycles 0101 Filter glitches up to width of 5 IPS clock cycles 0110 Filter glitches up to width of 6 IPS clock cycles 0111 Filter glitches up to width of 7 IPS clock cycles 1000 Filter glitches up to width of 8 IPS clock cycles 1001 Filter glitches up to width of 9 IPS clock cycles 1010 Filter glitches up to width of 10 IPS clock cycles 1011 Filter glitches up to width of 11 IPS clock cycles 1100 Filter glitches up to width of 12 IPS clock cycles 1101 Filter glitches up to width of 13 IPS clock cycles 1110 Filter glitches up to width of 14 IPS clock cycles 1111 Filter glitches up to width of 15 IPS clock cycles

This filter can absorb glitches on the I²C clock and data lines for each I²C module. The width of the glitch to absorb is specified in terms of number of IPS clock cycles. This glitch filter control register is provided for all three I²C modules.

The programming of the glitch filter is simple; the programmer only needs to specify the size of glitch (in terms of IPS cycles) for the filter to absorb and not pass.

21.4 Initialization Sequence

Reset puts the I²C control register to its default status. Before the interface can transfer serial data, the following initialization procedure must be done:

Update the frequency divider register and select the required division ratio to obtain the SCL frequency from the IPS clock.

1. Calculate the divider according to the ips clock frequency and the expected scl frequency:

$$\text{scl_divider} = \text{f_IPS} / \text{f_SCL}.$$
2. Look up the table 39-6 I2C Frequency Divider Bit Selection to find out value of FDR bit file in MFDR register according to the calculated divider.
3. Set the prescaler in frequency divider register equal to FDR obtain from look-up table.

Update the I²C address register to define a slave address.

Set the control register EN bit to enable the I²C interface system.

Modify the control register bits to select master/slave mode, transmit/receive mode, and interrupt enable or not.

21.5 Transfer Initiation and Interrupt

In master transmit mode, a data transfer is initiated when data is written to the DATA register. The most significant bit is sent first.

In master receive mode, reading this register initiates next byte data receiving.

In slave mode, the same functions are available after an address match occurs. Data transfer is initiated by:

- Writing to the DATA register for slave transmits

or

- A dummy reading from the DATA register in slave receive mode occurs.

The I²C interrupt STATUS register bit is set when an interrupt is pending. If the CONTROL register interrupt enable bit is set, the I²C interrupt STATUS register bit, if set, causes a processor interrupt request. The interrupt bit sets when one of the following events occurs:

- A complete 1-Byte transfer (set at falling edge of ninth clock) occurs.
- A receive calling address matches its own specific address in slave receive mode.
- Arbitration is lost.

21.5.1 Post-Transfer Software Response

In the interrupt service routine, software must clear the IF status bit first. The CF status bit is cleared automatically by reading from the data I/O register (MDR) in receive mode or writing to MBDR in transmit mode.

Software may service the bus I/O in the main program by monitoring the IF status bit if the interrupt function is disabled. Polling should monitor the IF status bit rather than the CF bit because its operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit in the DATA register, the TX control bit should be toggled at this stage.

During slave mode address cycles (AAS = 1), the SRW bit in the STATUS register is read to determine the direction of the subsequent transfer and the TX control bit is programmed accordingly. The SRW bit is not valid for data cycles (AAS = 0) when operating in slave mode. Therefore, the TX bit in the control register should be read to determine the direction of the current transfer.

21.5.2 Slave Mode

In the slave interrupt service routine, the AAS bit should be tested to determine if a calling of its own address was received. If AAS is set, software should set the Tx/Rx mode select bit (control register Tx bit) according to the R/\overline{W} command bit (SRW). Writing to the CONTROL register automatically clears AAS. The slave interrupt service routine should also move the data, depending on whether it acts as a transmitter or a receiver, as follows:

- For a slave transmitter, the slave interrupt service routine must initiate a data transfer by writing information to the DATA register.
- For a slave receiver, the slave interrupt service routine must initiate a transfer by performing a dummy read from the DATA register. The slave drives SCL low between byte transfers. SCL is released when the DATA register is accessed in the required mode.

In slave transmitter routine, RXAK must be tested before transmitting the next data byte. Setting RXAK means an end of data signal from the master receiver. Then, software causes a switch from transmitter mode to receiver mode. A dummy read then releases the SCL line letting the master generate a STOP signal.

21.5.3 Special Note on AKF

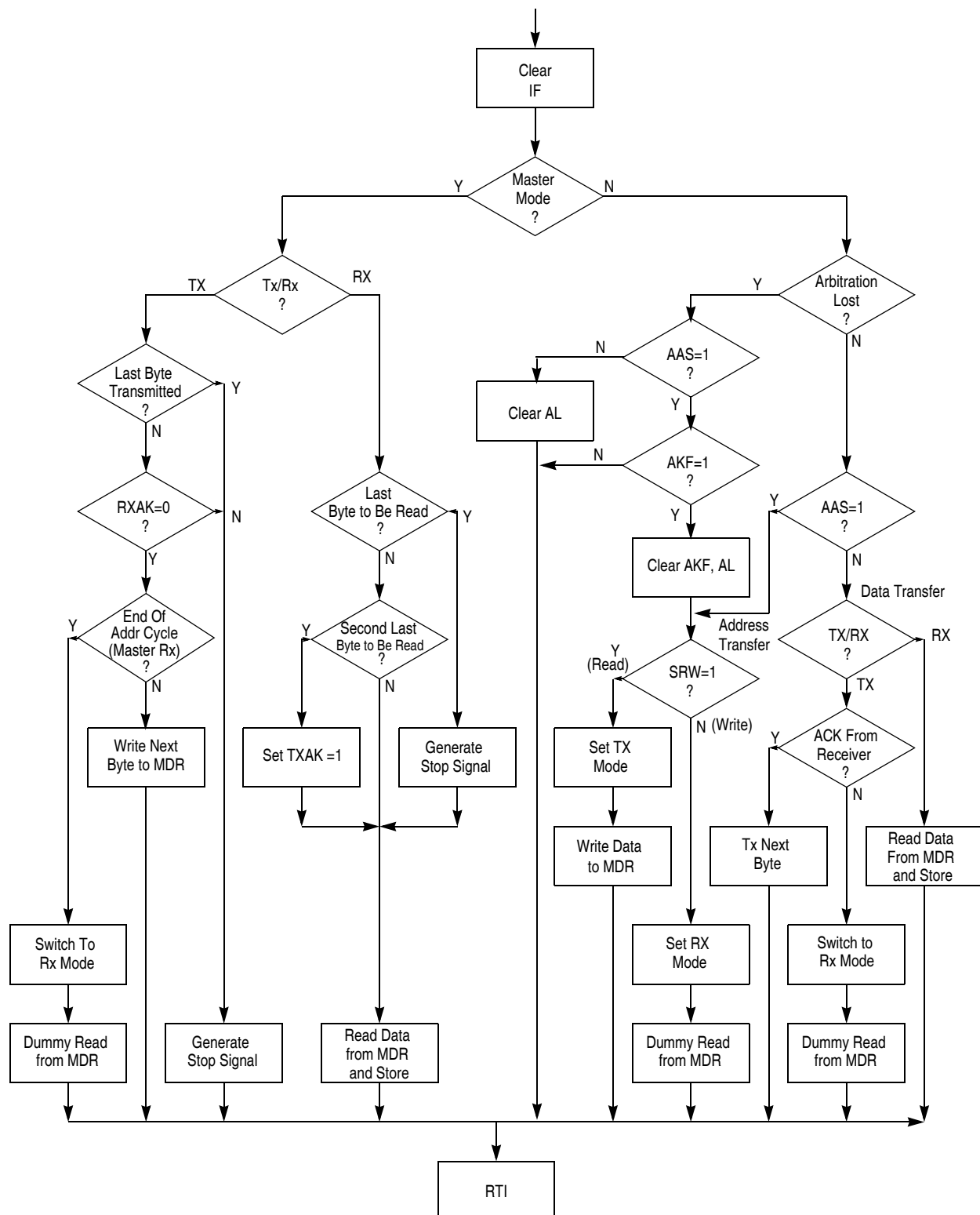
When the I²C module loses arbitration(AL) and is addressed as slave(AAS), the I²C module generates two interrupt (IF) requests: one on the rising edge of the acknowledge clock pulse and one on the falling edge of the acknowledge clock pulse (the legacy time to do so). In this specific case, the interrupt (IF=1) is really the second one set by the hardware.

The software programmer must use this AKF bit to distinguish the first interrupt request (non-legacy) from the second interrupt request (legacy) and may use the value of AKF (in addition to all the usual bits the software checks) to determine when to take action for the AL and AAS.

- First interrupt on rising edge of acknowledge cycle — software should not take AL and AAS action
- Second interrupt on falling edge of acknowledge — software should take AL and AAS action

The AKF bit is set only for the second interrupt and must be cleared by software writing it low in the interrupt routine.

[Figure 21-16](#) on typical software flow for I²C routines clearly illustrates how the AKF bit may be used.

Figure 21-16. Software Flowchart of Typical I²C Interrupt Routine

Chapter 22

IO Control

22.1 Introduction

22.1.1 Overview

The IO control block controls the functional muxing and configuration of the pads. Configurable parameters include slew rate, Schmitt-Trigger input, pull-down/-up, and PCI hold timing. Additionally, the IO control block implements a fifth layer of pin muxing between the GPTimer[7:0] and GPIO[7:0].

22.1.2 Features

- Functional Pin muxing control
 - Control for the IO macros
 - Muxing of GPTimer with GPIO
- Pad Slewrate control
- Pad Schmitt-Trigger control
- Pad Pull-down/-up control
- PCI timing delay cell control

22.2 Memory Map and Register Definition

22.2.1 Memory Map

Table 22-1. IO Control Memory Map (Sheet 1 of 7)

Offset or Address	Register	Access
General Registers		
0x000	IO_CONTROL_MEM -- MEM pad control register	R/W
0x004	IO_CONTROL_GP -- GP pad control register	R/W
0x008	IO_CONTROL_LPC_CLK -- LPC_CLK pad control register	R/W
0x00C	IO_CONTROL_LPC_OE -- $\overline{\text{LPC_OE}}$ pad control register	R/W
0x010	IO_CONTROL_LPC_R/W -- $\overline{\text{LPC_R/W}}$ pad control register	R/W
0x014	IO_CONTROL_LPC_ACK -- $\overline{\text{LPC_ACK}}$ pad control register	R/W
0x018	IO_CONTROL_LPC_CS0 -- $\overline{\text{LPC_CS0}}$ pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 2 of 7)

Offset or Address	Register	Access
0x01C	IO_CONTROL_NFC_CE0 -- NFC_CE0 pad control register	R/W
0x020	IO_CONTROL_LPC_CS1 -- LPC_CS1 pad control register	R/W
0x024	IO_CONTROL_LPC_CS2 -- LPC_CS2 pad control register	R/W
0x028	IO_CONTROL_LPC_AX03 -- LPC_AX03 pad control register	R/W
0x02C	IO_CONTROL_EMB_AX02 -- EMB_AX02 pad control register	R/W
0x030	IO_CONTROL_EMB_AX01 -- EMB_AX01 pad control register	R/W
0x034	IO_CONTROL_EMB_AX00 -- EMB_AX00 pad control register	R/W
0x038	IO_CONTROL_EMB_AD31 -- EMB_AD31 pad control register	R/W
0x03C	IO_CONTROL_EMB_AD30 -- EMB_AD30 pad control register	R/W
0x040	IO_CONTROL_EMB_AD29 -- EMB_AD29 pad control register	R/W
0x044	IO_CONTROL_EMB_AD28 -- EMB_AD28 pad control register	R/W
0x048	IO_CONTROL_EMB_AD27 -- EMB_AD27 pad control register	R/W
0x04C	IO_CONTROL_EMB_AD26 -- EMB_AD26 pad control register	R/W
0x050	IO_CONTROL_EMB_AD25 -- EMB_AD25 pad control register	R/W
0x054	IO_CONTROL_EMB_AD24 -- EMB_AD24 pad control register	R/W
0x058	IO_CONTROL_EMB_AD23 -- EMB_AD23 pad control register	R/W
0x05C	IO_CONTROL_EMB_AD22 -- EMB_AD22 pad control register	R/W
0x060	IO_CONTROL_EMB_AD21 -- EMB_AD21 pad control register	R/W
0x064	IO_CONTROL_EMB_AD20 -- EMB_AD20 pad control register	R/W
0x068	IO_CONTROL_EMB_AD19 -- EMB_AD19 pad control register	R/W
0x06C	IO_CONTROL_EMB_AD18 -- EMB_AD18 pad control register	R/W
0x070	IO_CONTROL_EMB_AD17 -- EMB_AD17 pad control register	R/W
0x074	IO_CONTROL_EMB_AD16 -- EMB_AD16 pad control register	R/W
0x078	IO_CONTROL_EMB_AD15 -- EMB_AD15 pad control register	R/W
0x07C	IO_CONTROL_EMB_AD14 -- EMB_AD14 pad control register	R/W
0x080	IO_CONTROL_EMB_AD13 -- EMB_AD13 pad control register	R/W
0x084	IO_CONTROL_EMB_AD12 -- EMB_AD12 pad control register	R/W
0x088	IO_CONTROL_EMB_AD11 -- EMB_AD11 pad control register	R/W
0x08C	IO_CONTROL_EMB_AD10 -- EMB_AD10 pad control register	R/W
0x090	IO_CONTROL_EMB_AD09 -- EMB_AD09 pad control register	R/W
0x094	IO_CONTROL_EMB_AD08 -- EMB_AD08 pad control register	R/W
0x098	IO_CONTROL_EMB_AD07 -- EMB_AD07 pad control register	R/W
0x09C	IO_CONTROL_EMB_AD06 -- EMB_AD06 pad control register	R/W
0x0A0	IO_CONTROL_EMB_AD05 -- EMB_AD05 pad control register	R/W
0x0A4	IO_CONTROL_EMB_AD04 -- EMB_AD04 pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 3 of 7)

Offset or Address	Register	Access
0x0A8	IO_CONTROL_EMB_AD03 -- EMB_AD03 pad control register	R/W
0x0AC	IO_CONTROL_EMB_AD02 -- EMB_AD02 pad control register	R/W
0x0B0	IO_CONTROL_EMB_AD01 -- EMB_AD01 pad control register	R/W
0x0B4	IO_CONTROL_EMB_AD00 -- EMB_AD00 pad control register	R/W
0x0B8	IO_CONTROL_PATA_CE1 -- PATA_CE1 pad control register	R/W
0x0BC	IO_CONTROL_PATA_CE2 -- PATA_CE2 pad control register	R/W
0x0C0	IO_CONTROL_PATA_ISOLATE -- PATA_ISOLATE pad control register	R/W
0x0C4	IO_CONTROL_PATA_IOR -- PATA_IOR pad control register	R/W
0x0C8	IO_CONTROL_PATA_IOW -- PATA_IOW pad control register	R/W
0x0CC	IO_CONTROL_PATA_IOCHRDY -- PATA_IOCHRDY pad control register	R/W
0x0D0	IO_CONTROL_PATA_INTRQ -- PATA_INTRQ pad control register	R/W
0x0D4	IO_CONTROL_PATA_DRQ -- PATA_DRQ pad control register	R/W
0x0D8	IO_CONTROL_PATA_DACK -- PATA_DACK pad control register	R/W
0x0DC	IO_CONTROL_NFC_WP -- NFC_WP pad control register	R/W
0x0E0	IO_CONTROL_NFC_RB -- NFC_RB pad control register	R/W
0x0E4	IO_CONTROL_NFC_ALE -- NFC_ALE pad control register	R/W
0x0E8	IO_CONTROL_NFC_CLE -- NFC_CLE pad control register	R/W
0x0EC	IO_CONTROL_NFC_WE -- NFC_WE pad control register	R/W
0x0F0	IO_CONTROL_NFC_RE -- NFC_RE pad control register	R/W
0x0F4	IO_CONTROL_PCI_AD31 -- PCI_AD31 pad control register	R/W
0x0F8	IO_CONTROL_PCI_AD30 -- PCI_AD30 pad control register	R/W
0x0FC	IO_CONTROL_PCI_AD29 -- PCI_AD29 pad control register	R/W
0x100	IO_CONTROL_PCI_AD28 -- PCI_AD28 pad control register	R/W
0x104	IO_CONTROL_PCI_AD27 -- PCI_AD27 pad control register	R/W
0x108	IO_CONTROL_PCI_AD26 -- PCI_AD26 pad control register	R/W
0x10C	IO_CONTROL_PCI_AD25 -- PCI_AD25 pad control register	R/W
0x110	IO_CONTROL_PCI_AD24 -- PCI_AD24 pad control register	R/W
0x114	IO_CONTROL_PCI_AD23 -- PCI_AD23 pad control register	R/W
0x118	IO_CONTROL_PCI_AD22 -- PCI_AD22 pad control register	R/W
0x11C	IO_CONTROL_PCI_AD21 -- PCI_AD21 pad control register	R/W
0x120	IO_CONTROL_PCI_AD20 -- PCI_AD20 pad control register	R/W
0x124	IO_CONTROL_PCI_AD19 -- PCI_AD19 pad control register	R/W
0x128	IO_CONTROL_PCI_AD18 -- PCI_AD18 pad control register	R/W
0x12C	IO_CONTROL_PCI_AD17 -- PCI_AD17 pad control register	R/W
0x130	IO_CONTROL_PCI_AD16 -- PCI_AD16 pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 4 of 7)

Offset or Address	Register	Access
0x134	IO_CONTROL_PCI_AD15 -- PCI_AD15 pad control register	R/W
0x138	IO_CONTROL_PCI_AD14 -- PCI_AD14 pad control register	R/W
0x13C	IO_CONTROL_PCI_AD13 -- PCI_AD13 pad control register	R/W
0x140	IO_CONTROL_PCI_AD12 -- PCI_AD12 pad control register	R/W
0x144	IO_CONTROL_PCI_AD11 -- PCI_AD11 pad control register	R/W
0x148	IO_CONTROL_PCI_AD10 -- PCI_AD10 pad control register	R/W
0x14C	IO_CONTROL_PCI_AD09 -- PCI_AD09 pad control register	R/W
0x150	IO_CONTROL_PCI_AD08 -- PCI_AD08 pad control register	R/W
0x154	IO_CONTROL_PCI_AD07 -- PCI_AD07 pad control register	R/W
0x158	IO_CONTROL_PCI_AD06 -- PCI_AD06 pad control register	R/W
0x15C	IO_CONTROL_PCI_AD05 -- PCI_AD05 pad control register	R/W
0x160	IO_CONTROL_PCI_AD04 -- PCI_AD04 pad control register	R/W
0x164	IO_CONTROL_PCI_AD03 -- PCI_AD03 pad control register	R/W
0x168	IO_CONTROL_PCI_AD02 -- PCI_AD02 pad control register	R/W
0x16C	IO_CONTROL_PCI_AD01 -- PCI_AD01 pad control register	R/W
0x170	IO_CONTROL_PCI_AD00 -- PCI_AD00 pad control register	R/W
0x174	IO_CONTROL_PCI_CBE0 -- PCI_CBE0 pad control register	R/W
0x178	IO_CONTROL_PCI_CBE1 -- PCI_CBE1 pad control register	R/W
0x17C	IO_CONTROL_PCI_CBE2 -- PCI_CBE2 pad control register	R/W
0x180	IO_CONTROL_PCI_CBE3 -- PCI_CBE3 pad control register	R/W
0x184	IO_CONTROL_PCI_GRANT2 -- PCI_GRANT2 pad control register	R/W
0x188	IO_CONTROL_PCI_REQ2 -- PCI_REQ2 pad control register	R/W
0x18C	IO_CONTROL_PCI_GRANT1 -- PCI_GRANT1 pad control register	R/W
0x190	IO_CONTROL_PCI_REQ1 -- PCI_REQ1 pad control register	R/W
0x194	IO_CONTROL_PCI_GRANT0 -- PCI_GRANT0 pad control register	R/W
0x198	IO_CONTROL_PCI_REQ0 -- PCI_REQ0 pad control register	R/W
0x19C	IO_CONTROL_PCI_INTA -- PCI_INTA pad control register	R/W
0x1A0	IO_CONTROL_PCI_CLK -- PCI_CLK pad control register	R/W
0x1A4	IO_CONTROL_PCI_RST -- PCI_RST pad control register	R/W
0x1A8	IO_CONTROL_PCI_FRAME -- PCI_FRAME pad control register	R/W
0x1AC	IO_CONTROL_PCI_IDSEL -- PCI_IDSEL pad control register	R/W
0x1B0	IO_CONTROL_PCI_DEVSEL-PCI_DEVSEL pad control register	R/W
0x1B4	IO_CONTROL_PCI_IRDY -- PCI_IRDY pad control register	R/W
0x1B8	IO_CONTROL_PCI_TRDY -- PCI_TRDY pad control register	R/W
0x1BC	IO_CONTROL_PCI_STOP -- PCI_STOP pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 5 of 7)

Offset or Address	Register	Access
0x1C0	IO_CONTROL_PCI_PAR -- PCI_PAR pad control register	R/W
0x1C4	IO_CONTROL_PCI_PERR -- PCI_PERR pad control register	R/W
0x1C8	IO_CONTROL_PCI_SERR -- PCI_SERR pad control register	R/W
0x1CC	IO_CONTROL_SPDIF_TXCLK -- SPDIF_TXCLK pad control register	R/W
0x1D0	IO_CONTROL_SPDIF_TX -- SPDIF_TX pad control register	R/W
0x1D4	IO_CONTROL_SPDIF_RX -- SPDIF_RX pad control register	R/W
0x1D8	IO_CONTROL_I2C0_SCL -- I2C0_SCL pad control register	R/W
0x1DC	IO_CONTROL_I2C0_SDA -- I2C0_SDA pad control register	R/W
0x1E0	IO_CONTROL_I2C1_SCL -- I2C1_SCL pad control register	R/W
0x1E4	IO_CONTROL_I2C1_SDA -- I2C1_SDA pad control register	R/W
0x1E8	IO_CONTROL_I2C2_SCL -- I2C2_SCL pad control register	R/W
0x1EC	IO_CONTROL_I2C2_SDA -- I2C2_SDA pad control register	R/W
0x1F0	IO_CONTROL_IRQ0 -- IRQ0 pad control register	R/W
0x1F4	IO_CONTROL_IRQ1 -- IRQ1 pad control register	R/W
0x1F8	IO_CONTROL_CAN1_TX -- CAN1_TX pad control register	R/W
0x1FC	IO_CONTROL_CAN2_TX -- CAN2_TX pad control register	R/W
0x200	IO_CONTROL_J1850_TX -- J1850_TX pad control register	R/W
0x204	IO_CONTROL_J1850_RX -- J1850_RX pad control register	R/W
0x208	IO_CONTROL_PSC_MCLK_IN -- PSC_MCLK_IN pad control register	R/W
0x20C	IO_CONTROL_PSC0_0 -- PSC0_0 pad control register	R/W
0x210	IO_CONTROL_PSC0_1 -- PSC0_1 pad control register	R/W
0x214	IO_CONTROL_PSC0_2 -- PSC0_2 pad control register	R/W
0x218	IO_CONTROL_PSC0_3 -- PSC0_3 pad control register	R/W
0x21C	IO_CONTROL_PSC0_4 -- PSC0_4 pad control register	R/W
0x220	IO_CONTROL_PSC1_0 -- PSC1_0 pad control register	R/W
0x224	IO_CONTROL_PSC1_1 -- PSC1_1 pad control register	R/W
0x228	IO_CONTROL_PSC1_2 -- PSC1_2 pad control register	R/W
0x22C	IO_CONTROL_PSC1_3 -- PSC1_3 pad control register	R/W
0x230	IO_CONTROL_PSC1_4 -- PSC1_4 pad control register	R/W
0x234	IO_CONTROL_PSC2_0 -- PSC2_0 pad control register	R/W
0x238	IO_CONTROL_PSC2_1 -- PSC2_1 pad control register	R/W
0x23C	IO_CONTROL_PSC2_2 -- PSC2_2 pad control register	R/W
0x240	IO_CONTROL_PSC2_3 -- PSC2_3 pad control register	R/W
0x244	IO_CONTROL_PSC2_4 -- PSC2_4 pad control register	R/W
0x248	IO_CONTROL_PSC3_0 -- PSC3_0 pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 6 of 7)

Offset or Address	Register	Access
0x24C	IO_CONTROL_PSC3_1 -- PSC3_1 pad control register	R/W
0x250	IO_CONTROL_PSC3_2 -- PSC3_2 pad control register	R/W
0x254	IO_CONTROL_PSC3_3 -- PSC3_3 pad control register	R/W
0x258	IO_CONTROL_PSC3_4 -- PSC3_4 pad control register	R/W
0x25C	IO_CONTROL_PSC4_0 -- PSC4_0 pad control register	R/W
0x260	IO_CONTROL_PSC4_1 -- PSC4_1 pad control register	R/W
0x264	IO_CONTROL_PSC4_2 -- PSC4_2 pad control register	R/W
0x268	IO_CONTROL_PSC4_3 -- PSC4_3 pad control register	R/W
0x26C	IO_CONTROL_PSC4_4 -- PSC4_4 pad control register	R/W
0x270	IO_CONTROL_PSC5_0 -- PSC5_0 pad control register	R/W
0x274	IO_CONTROL_PSC5_1 -- PSC5_1 pad control register	R/W
0x278	IO_CONTROL_PSC5_2 -- PSC5_2 pad control register	R/W
0x27C	IO_CONTROL_PSC5_3 -- PSC5_3 pad control register	R/W
0x280	IO_CONTROL_PSC5_4 -- PSC5_4 pad control register	R/W
0x284	IO_CONTROL_PSC6_0 -- PSC6_0 pad control register	R/W
0x288	IO_CONTROL_PSC6_1 -- PSC6_1 pad control register	R/W
0x28C	IO_CONTROL_PSC6_2 -- PSC6_2 pad control register	R/W
0x290	IO_CONTROL_PSC6_3 -- PSC6_3 pad control register	R/W
0x294	IO_CONTROL_PSC6_4 -- PSC6_4 pad control register	R/W
0x298	IO_CONTROL_PSC7_0 -- PSC7_0 pad control register	R/W
0x29C	IO_CONTROL_PSC7_1 -- PSC7_1 pad control register	R/W
0x2A0	IO_CONTROL_PSC7_2 -- PSC7_2 pad control register	R/W
0x2A4	IO_CONTROL_PSC7_3 -- PSC7_3 pad control register	R/W
0x2A8	IO_CONTROL_PSC7_4 -- PSC7_4 pad control register	R/W
0x2AC	IO_CONTROL_PSC8_0 -- PSC8_0 pad control register	R/W
0x2B0	IO_CONTROL_PSC8_1 -- PSC8_1 pad control register	R/W
0x2B4	IO_CONTROL_PSC8_2 -- PSC8_2 pad control register	R/W
0x2B8	IO_CONTROL_PSC8_3 -- PSC8_3 pad control register	R/W
0x2BC	IO_CONTROL_PSC8_4 -- PSC8_4 pad control register	R/W
0x2C0	IO_CONTROL_PSC9_0 -- PSC9_0 pad control register	R/W
0x2C4	IO_CONTROL_PSC9_1 -- PSC9_1 pad control register	R/W
0x2C8	IO_CONTROL_PSC9_2 -- PSC9_2 pad control register	R/W
0x2CC	IO_CONTROL_PSC9_3 -- PSC9_3 pad control register	R/W
0x2D0	IO_CONTROL_PSC9_4 -- PSC9_4 pad control register	R/W
0x2D4	IO_CONTROL_PSC10_0 -- PSC10_0 pad control register	R/W

Table 22-1. IO Control Memory Map (Sheet 7 of 7)

Offset or Address	Register	Access
0x2D8	IO_CONTROL_PSC10_1 -- PSC10_1 pad control register	R/W
0x2DC	IO_CONTROL_PSC10_2 -- PSC10_2 pad control register	R/W
0x2E0	IO_CONTROL_PSC10_3 -- PSC10_3 pad control register	R/W
0x2E4	IO_CONTROL_PSC10_4 -- PSC10_4 pad control register	R/W
0x2E8	IO_CONTROL_PSC11_0 -- PSC11_0 pad control register	R/W
0x2EC	IO_CONTROL_PSC11_1 -- PSC11_1 pad control register	R/W
0x2F0	IO_CONTROL_PSC11_2 -- PSC11_2 pad control register	R/W
0x2F4	IO_CONTROL_PSC11_3 -- PSC11_3 pad control register	R/W
0x2F8	IO_CONTROL_PSC11_4 -- PSC11_4 pad control register	R/W
0x304	IO_CONTROL_CKSTP_OUT -- CKSTP_OUT pad control register	R/W
0x310	IO_CONTROL_USB_PHY_DRVVBUS -- USB2_DRVVBUS pad control register	R/W

22.2.2 Register Descriptions

22.2.2.1 IO_CONTROL_MEM Register

Offset 0x000								Access: User read/write								
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	16BIT	CONT_DS			DATA_DS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		= Unimplemented or Reserved														

Figure 22-1. MEM IO Control Register (IO_CONTROL_MEM)

Table 22-2. IO_CONTROL_MEM Field Descriptions

Field	Description
16BIT	Enables pin muxing in DRAM 16 bit mode. 0 All DDR pads are used for DRAM functionality 1 MDQ[31:16], MDM[3:2] and MDQS[3:2] have GPIO functionality
CONT_DS	CONT_DS controls the slew rate/drive strength of all DDR control pads (MCS, MA[15:0], MCK, $\overline{\text{MCK}}$, MODT, MCAS, MRAS, MCKE, MWE, and MBA[2:0]). 000 DDR pad configuration 0 001 DDR pad configuration 1 010 DDR pad configuration 2 011 DDR pad configuration 3 100 Reserved 101 Reserved 110 DDR pad configuration 6 111 Reserved Note: DDR pad configurations are defined in the Datasheet
DATA_DS	DATA_DS controls the slew rate/drive strength of all DDR data pads (MDQ[31:0], MDM[3:0], and MDQS[3:0]). 000 DDR pad configuration 0 001 DDR pad configuration 1 010 DDR pad configuration 2 011 DDR pad configuration 3 100 Reserved 101 Reserved 110 DDR pad configuration 6 111 Reserved Note: The configured DDR data pads is also valid for the GPIO lines in 16BIT mode configuration. Note: DDR pad configurations are defined in the Datasheet

22.2.2.2 IO_CONTROL_GP Register

Offset 0x004								Access: User read/write								
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	GP_MUX0	GP_MUX1	GP_MUX2	GP_MUX3	GP_MUX4	GP_MUX5	GP_MUX6	GP_MUX7
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 22-2. GP MUX Control Register (IO_CONTROL_GP)

Table 22-3. IO_CONTROL_GP Field Descriptions

Field	Description
GP_MUX7	GP_MUX7 controls the muxing between GPIO7 and GPT7. 0 GPT7 is selected 1 GPIO7 is selected
GP_MUX6	GP_MUX6 controls the muxing between GPIO6 and GPT6. 0 GPT6 is selected 1 GPIO6 is selected
GP_MUX5	GP_MUX5 controls the muxing between GPIO5 and GPT5. 0 GPT5 is selected 1 GPIO5 is selected
GP_MUX4	GP_MUX4 controls the muxing between GPIO4 and GPT4. 0 GPT4 is selected 1 GPIO4 is selected
GP_MUX3	GP_MUX3 controls the muxing between GPIO3 and GPT3. 0 GPT3 is selected 1 GPIO3 is selected
GP_MUX2	GP_MUX2 controls the muxing between GPIO2 and GPT2. 0 GPT2 is selected 1 GPIO2 is selected

Table 22-3. IO_CONTROL_GP Field Descriptions (continued)

GP_MUX1	GP_MUX1 controls the muxing between GPIO1 and GPT1. 0 GPT1 is selected 1 GPIO1 is selected
GP_MUX0	GP_MUX0 controls the muxing between GPIO0 and GPT0. 0 GPT0 is selected 1 GPIO0 is selected

22.2.2.3 IO_CONTROL_PAD Registers Descriptions

There are six different types of IO_CONTROL_PAD registers. With all register it is possible to configure the slew rate of the pad and to configure the functional muxing. The following different types are used:

- STD
 - functional muxing
 - programmable slew rate
- STD_PU
 - functional muxing
 - programmable slew rate
 - programmable Pull-up/down resistors
- STD_ST
 - functional muxing
 - programmable slew rate
 - programmable Schmitt-Trigger input
- STD_PU_ST
 - functional muxing
 - programmable slew rate
 - programmable Schmitt-Trigger input
 - programmable Pull-up/down resistors
- PCI
 - functional muxing
 - programmable slew rate
 - programmable PCI output delay timing
- PCI_ST
 - functional muxing
 - programmable slew rate
 - programmable PCI output delay timing
 - programmable Schmitt-Trigger input

Figure 22-3, Figure 22-4, Figure 22-5, Figure 22-6, Figure 22-7, and Figure 22-8 describe the different type of configuration registers. The Table 22-16 shows which type of configuration register is responsible to configure the different pads. Additionally it shows also the different functional muxing possibilities.

22.2.2.3.1 Standard (STD)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX			0	0	0	0	0	DS
W																
Reset	0	0	0	0	0	0	0	-	-	0	0	0	0	0	-	-

Reset values of the register bits is described in [Table 22-16](#)

 = Unimplemented or Reserved

Figure 22-3. Standard Control Register (STD)

Table 22-4. STD Field Descriptions

Field	Description
FUNCMUX	<p>FUNCMUX controls the functional pin muxing of the pad.</p> <p>00 Function 1</p> <p>01 Function 2</p> <p>10 Function 3</p> <p>11 Function 4</p> <p>Note: Function assignment is listed in Table 22-16</p>
DS	<p>DS controls slew rate of General IO pad</p> <p>00 General IO slew rate configuration 0</p> <p>01 General IO slew rate configuration 1</p> <p>10 General IO slew rate configuration 2</p> <p>11 General IO slew rate configuration 3</p> <p>Note: General IO slew rate configurations are defined in the Datasheet</p>

Table 22-5.

22.2.2.3.2 Standard with Pull-up/down Resistors (STD_PU)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX		0	0	PUD	PUE	0	DS	
W																
Reset	0	0	0	0	0	0	0	-	-	0	0	-	-	0	-	-

Reset values of the register bits is described in [Table 22-16](#)


 = Unimplemented or Reserved

Figure 22-4. Standard with Pull-up/down Resistors Control Register (STD_PU)

Table 22-6. STD_PU Field Descriptions

Field	Description
FUNCMUX	<p>FUNCMUX controls the functional pin muxing of the pad.</p> <p>00 Function 1 01 Function 2 10 Function 3 11 Function 4</p> <p>Note: Function assignment is listed in Table 22-16</p>
PUD	<p>PUD controls the direction of the pull resistors.</p> <p>0 Pull-down resistor enabled, if PUE is 1 1 Pull-up resistor enabled, if PUE is 1</p>
PUE	<p>PUE enables the pull usage</p> <p>0 No Pull resistor is used 1 Pull resistor is enabled</p>
DS	<p>DS controls slew rate of General IO pad</p> <p>00 General IO slew rate configuration 0 01 General IO slew rate configuration 1 10 General IO slew rate configuration 2 11 General IO slew rate configuration 3</p> <p>Note: General IO slew rate configurations are defined in the Datasheet</p>

Table 22-7.

22.2.2.3.3 Standard with Schmitt-Trigger Input (STD_ST)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX			0	0	0	0	ST	DS
W																
Reset	0	0	0	0	0	0	0	-	-	0	0	0	0	-	-	-

Reset values of the register bits is described in [Table 22-16](#)

 = Unimplemented or Reserved

Figure 22-5. Standard with Schmitt-Trigger Input Control Register (STD_ST)

Table 22-8. STD_ST Field Descriptions

Field	Description
FUNCMUX	FUNCMUX controls the functional pin muxing of the pad. 00 Function 1 01 Function 2 10 Function 3 11 Function 4 Note: Function assignment is listed in Table 22-16
ST	ST enables the Schmitt Trigger input of the pad 0 Schmitt Trigger input disabled 1 Schmitt Trigger input enabled
DS	DS controls slew rate of General IO pad 00 General IO slew rate configuration 0 01 General IO slew rate configuration 1 10 General IO slew rate configuration 2 11 General IO slew rate configuration 3 Note: General IO slew rate configurations are defined in the Datasheet

Table 22-9.



**22.2.2.3.4 Standard with Pull-up/down Resistors and Schmitt-Trigger Input
(STD_PU_ST)**

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

Table 22-11.

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX		0	0	PUD		PUE	ST	DS
W																
Reset	0	0	0	0	0	0	-	-	0	0	-	-	-	-	-	-

Reset values of the register bits is described in [Table 22-16](#)

 = Unimplemented or Reserved

Figure 22-6. Standard with Pull-up/down Resistors and Schmitt-Trigger Input Control Register (STD_PU_ST)

Table 22-10. STD_PU_ST Field Descriptions

Field	Description
FUNCMUX	FUNCMUX controls the functional pin muxing of the pad. 00 Function 1 01 Function 2 10 Function 3 11 Function 4 Note: Function assignment is listed in Table 22-16
PUD	PUD controls the direction of the pull resistors. 0 Pull-down resistor enabled, if PUE is 1 1 Pull-up resistor enabled, if PUE is 1
PUE	PUE enables the pull usage 0 No Pull resistor is used 1 Pull resistor is enabled
ST	ST enables the Schmitt Trigger input of the pad 0 Schmitt Trigger input disabled 1 Schmitt Trigger input enabled
DS	DS controls slew rate of General IO pad 00 General IO slew rate configuration 0 01 General IO slew rate configuration 1 10 General IO slew rate configuration 2 11 General IO slew rate configuration 3 Note: General IO slew rate configurations are defined in the Datasheet

Table 22-11.

22.2.2.3.5 PCI (PCI)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX		HOLD		0	0	0	0	DS
W																
Reset	0	0	0	0	0	0	0	-	-	-	-	0	0	0	0	-

Reset values of the register bits is described in [Table 22-16](#)


 = Unimplemented or Reserved

Figure 22-7. PCI Control Register (PCI)

Table 22-12. PCI Field Descriptions

Field	Description
FUNCMUX	<p>FUNCMUX controls the functional pin muxing of the pad.</p> <p>00 Function 1 01 Function 2 10 Function 3 11 Function 4</p> <p>Note: Function assignment is listed in Table 22-16</p>
HOLD	<p>HOLD controls the PCI output hold delay</p> <p>00 PCI pad hold time configuration 0 01 PCI pad hold time configuration 1 10 PCI pad hold time configuration 2 11 PCI pad hold time configuration 3</p> <p>Note: PCI pad hold time configuration are defined in the Datasheet</p>
DS	<p>DS controls slew rate of PCI pad</p> <p>0 PCI slew rate configuration 0 1 PCI slew rate configuration 1</p> <p>Note: PCI slew rate configurations are defined in the Datasheet</p>

Table 22-13.

22.2.2.3.6 PCI with Schmitt-Trigger input (PCI_ST)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	FUNCMUX		HOLD		0	0	ST	0	DS
W																
Reset	0	0	0	0	0	0	0	-	-	-	-	0	0	-	0	-

Reset values of the register bits is described in [Table 22-16](#)

 = Unimplemented or Reserved

Figure 22-8. PCI with Schmitt-Trigger Input Control Register (PCI_ST)

Table 22-14. PCI_ST Field Descriptions

Field	Description
FUNCMUX	FUNCMUX controls the functional pin muxing of the pad. 00 Function 1 01 Function 2 10 Function 3 11 Function 4 Note: Function assignment is listed in Table 22-16
HOLD	HOLD controls the PCI output hold delay 00 PCI pad hold time configuration 0 01 PCI pad hold time configuration 1 10 PCI pad hold time configuration 2 11 PCI pad hold time configuration 3 Note: PCI pad hold time configuration are defined in the Datasheet
ST	ST enables the Schmitt Trigger input of the pad 0 Schmitt Trigger input disabled 1 Schmitt Trigger input enabled
DS	DS controls slew rate of PCI pad 0 PCI slew rate configuration 0 1 PCI slew rate configuration 1 Note: PCI slew rate configurations are defined in the Datasheet

Table 22-15.

22.2.2.3.7 Pad IO Control Register

Table 22-16. Pad IO Control Register Table (Sheet 1 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
LPC_CLK	0x008	STD	0_0000_00?? ¹	00 - LPC_CLK 01 - TPA 10 - CKSTP_IN 11 - Reserved
LPC_OE	0x00C	STD	0_0000_00?? ¹	00 - LPC_OE 01 - Reserved 10 - Reserved 11 - Reserved
LPC_RW	0x010	STD	0_0000_00?? ¹	00 - LPC_RW 01 - Reserved 10 - Reserved 11 - Reserved
LPC_ACK	0x014	STD_PU	0_0001_10?? ¹	00 - LPC_ACK 01 - LPC_CS ₇ 10 - Reserved 11 - GPIO24
LPC_CS0	0x018	STD_PU	0_0001_10?? ¹	00 - LPC_CS0 01 - Reserved 10 - Reserved 11 - GPIO25
NFC_CE0	0x01C	STD_PU_ST	0_0001_10?? ²	00 - NFC_CE0 01 - LPC_CS3 10 - PSC_MCLK_IN 11 - GPIO26
LPC_CS1	0x020	STD_PU_ST	0_0001_1000	00 - LPC_CS1 01 - SPDIF_TXCLK 10 - Reserved 11 - GPIO7
LPC_CS2	0x024	STD_PU	0_0001_1000	00 - LPC_CS2 01 - NFC_CE1 10 - Reserved 11 - GPIO0
LPC_AX03	0x028	STD	0_0000_00?? ¹	00 - LPC_AX03 01 - Reserved 10 - Reserved 11 - GPIO1
EMB_AX02	0x02C	STD	0_0000_00?? ¹	00 - EMB_AX02 01 - Reserved 10 - Reserved 11 - GPIO2
EMB_AX01	0x030	STD	0_0000_00?? ¹	00 - EMB_AX01 01 - Reserved 10 - Reserved 11 - GPIO3

Table 22-16. Pad IO Control Register Table (Sheet 2 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
EMB_AX00	0x034	STD	0_0000_00?? ¹	00 - EMB_AX00 01 - Reserved 10 - Reserved 11 - Reserved
EMB_AD31	0x038	STD	1_0000_00?? ²	00 - Reserved 01 - Reserved 10 - EMB_AD31 11 - Reserved
EMB_AD30	0x03C	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD30 11 - Reserved
EMB_AD29	0x040	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD29 11 - Reserved
EMB_AD28	0x044	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD28 11 - Reserved
EMB_AD27	0x048	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD27 11 - Reserved
EMB_AD26	0x04C	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD26 11 - Reserved
EMB_AD25	0x050	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD25 11 - Reserved
EMB_AD24	0x054	STD	1_0000_00?? ³	00 - Reserved 01 - Reserved 10 - EMB_AD24 11 - Reserved
EMB_AD23	0x058	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD23 11 - Reserved
EMB_AD22	0x05C	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD22 11 - Reserved

Table 22-16. Pad IO Control Register Table (Sheet 3 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
EMB_AD21	0x060	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD21 11 - Reserved
EMB_AD20	0x064	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD20 11 - Reserved
EMB_AD19	0x068	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD19 11 - Reserved
EMB_AD18	0x06C	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD18 11 - Reserved
EMB_AD17	0x070	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD17 11 - Reserved
EMB_AD16	0x074	STD	1_0000_0011	00 - Reserved 01 - Reserved 10 - EMB_AD16 11 - Reserved
EMB_AD15	0x078	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD15 11 - Reserved
EMB_AD14	0x07C	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD14 11 - Reserved
EMB_AD13	0x080	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD13 11 - Reserved
EMB_AD12	0x084	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD12 11 - Reserved
EMB_AD11	0x088	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD11 11 - Reserved

Table 22-16. Pad IO Control Register Table (Sheet 4 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
EMB_AD10	0x08C	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD10 11 - Reserved
EMB_AD09	0x090	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD09 11 - Reserved
EMB_AD08	0x094	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD08 11 - Reserved
EMB_AD07	0x098	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD07 11 - Reserved
EMB_AD06	0x09C	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD06 11 - Reserved
EMB_AD05	0x0A0	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD05 11 - Reserved
EMB_AD04	0x0A4	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD04 11 - Reserved
EMB_AD03	0x0A8	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD03 11 - Reserved
EMB_AD02	0x0AC	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD02 11 - Reserved
EMB_AD01	0x0B0	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD01 11 - Reserved
EMB_AD00	0x0B4	STD	1_0000_00?? ¹	00 - Reserved 01 - Reserved 10 - EMB_AD00 11 - Reserved

Table 22-16. Pad IO Control Register Table (Sheet 5 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PATA_CE1	0x0B8	STD_PU	0_0001_1000	00 - PATA_CE1 01 - LPC_CS4 10 - Reserved 11 - GPIO9
PATA_CE2	0x0BC	STD_PU	0_0001_1000	00 - PATA_CE2 01 - LPC_CS5 10 - Reserved 11 - GPIO10
PATA_ISOLATE	0x0C0	STD_PU	0_0001_1000	00 - PATA_ISOLATE 01 - CAN3_TX 10 - Reserved 11 - GPIO11
PATA_IOR	0x0C4	STD	0_0000_0000	00 - PATA_IOR 01 - SDHC_CLK 10 - Reserved 11 - GPIO12
PATA_IOW	0x0C8	STD_PU	?_0000_00?? ⁴	00 - PATA_IOW 01 - SDHC_CMD 10 - LPC_AX08 11 - GPIO13
PATA_IOCHRDY	0x0CC	STD_PU	?_0001_10?? ⁴	00 - PATA_IOCHRDY 01 - SDHC_D0 10 - LPC_AX07 11 - GPIO14
PATA_INTRQ	0x0D0	STD_PU	?_0000_10?? ⁴	00 - PATA_INTRQ 01 - SDHC_D1_IRQ 10 - LPC_AX06 11 - GPIO15
PATA_DRQ	0x0D4	STD_PU	?_0000_10?? ⁴	00 - PATA_DRQ 01 - SDHC_D2 10 - LPC_AX05 11 - GPIO16
PATA_DACK	0x0D8	STD_PU	?_0000_00?? ⁴	00 - PATA_DACK 01 - SDHC_D3_CD 10 - LPC_AX04 11 - GPIO17
NFC_WP	0x0DC	STD	?_0000_0011 ⁵	00 - NFC_WP 01 - SDHC_CLK 10 - LPC_AX09 11 - GPIO18
NFC_R/B	0x0E0	STD_PU	?_0001_10?? ⁷	00 - NFC_R/B 01 - SDHC_CMD 10 - LPC_AX08 11 - GPIO19

Table 22-16. Pad IO Control Register Table (Sheet 6 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
NFC_ALE	0x0E4	STD_PU	?_0000_0011 ⁵	00 - NFC_ALE 01 - SDHC_D0 10 - LPC_AX07 11 - GPIO20
NFC_CLE	0x0E8	STD_PU	?_0000_0011 ⁵	00 - NFC_CLE 01 - SDHC_D1_IRQ 10 - LPC_AX06 11 - GPIO21
NFC_WE	0x0EC	STD_PU	?_0000_0011 ⁵	00 - NFC_WE 01 - SDHC_D2 10 - LPC_AX05 11 - GPIO22
NFC_RE	0x0F0	STD_PU	?_0000_0011 ⁵	00 - NFC_RE 01 - SDHC_D3_CD 10 - LPC_AX04 11 - GPIO23
PCI_AD31	0x0F4	PCI_ST	0_0??0_0000 ⁶	00 - PCI_AD31 01 - USB0_CLK 10 - DIU_LD22 11 - GPIO0
PCI_AD30	0x0F8	PCI	0_0??0_0000 ⁶	00 - PCI_AD30 01 - USB0_DIR 10 - DIU_LD23 11 - GPIO1
PCI_AD29	0x0FC	PCI	0_0??0_0000 ⁶	00 - PCI_AD29 01 - Reserved 10 - USB1_DATA7 11 - GPIO2
PCI_AD28	0x100	PCI	0_0??0_0000 ⁶	00 - PCI_AD28 01 - Reserved 10 - USB1_DATA6 11 - GPIO3
PCI_AD27	0x104	PCI	0_0??0_0000 ⁶	00 - PCI_AD27 01 - Reserved 10 - USB1_DATA5 11 - GPIO4
PCI_AD26	0x108	PCI	0_0??0_0000 ⁶	00 - PCI_AD26 01 - Reserved 10 - USB1_DATA4 11 - GPIO5
PCI_AD25	0x10C	PCI	0_0??0_0000 ⁶	00 - PCI_AD25 01 - Reserved 10 - USB1_DATA3 11 - GPIO6

Table 22-16. Pad IO Control Register Table (Sheet 7 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PCI_AD24	0x110	PCI	0_0??0_0000 ⁶	00 - PCI_AD24 01 - Reserved 10 - USB1_DATA2 11 - GPIO7
PCI_AD23	0x114	PCI	0_0??0_0000 ⁶	00 - PCI_AD23 01 - Reserved 10 - USB1_DATA1 11 - GPIO8
PCI_AD22	0x118	PCI	0_0??0_0000 ⁶	00 - PCI_AD22 01 - Reserved 10 - USB1_DATA0 11 - GPIO9
PCI_AD21	0x11C	PCI	0_0??0_0000 ⁶	00 - PCI_AD21 01 - Reserved 10 - USB1_STOP 11 - GPIO10
PCI_AD20	0x120	PCI	0_0??0_0000 ⁶	00 - PCI_AD20 01 - Reserved 10 - USB1_NEXT 11 - GPIO11
PCI_AD19	0x124	PCI_ST	0_0??0_0000 ⁶	00 - PCI_AD19 01 - Reserved 10 - USB1_CLK 11 - GPIO12
PCI_AD18	0x128	PCI	0_0??0_0000 ⁶	00 - PCI_AD18 01 - Reserved 10 - USB1_DIR 11 - GPIO13
PCI_AD17	0x12C	PCI	0_0??0_0000 ⁶	00 - PCI_AD17 01 - VIU_DATA0 10 - FEC_TXD_3 11 - GPIO14
PCI_AD16	0x130	PCI	0_0??0_0000 ⁶	00 - PCI_AD16 01 - VIU_DATA1 10 - FEC_TXD_2 11 - GPIO15
PCI_AD15	0x134	PCI	0_0??0_0000 ⁶	00 - PCI_AD15 01 - VIU_DATA2 10 - FEC_TXD_1 11 - GPIO16
PCI_AD14	0x138	PCI	0_0??0_0000 ⁶	00 - PCI_AD14 01 - Reserved 10 - FEC_TXD_0 11 - GPIO17

Table 22-16. Pad IO Control Register Table (Sheet 8 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PCI_AD13	0x13C	PCI	0_0??0_0000 ⁶	00 - PCI_AD13 01 - VIU_DATA3 10 - FEC_RXD_3 11 - GPIO18
PCI_AD12	0x140	PCI	0_0??0_0000 ⁶	00 - PCI_AD12 01 - VIU_DATA4 10 - FEC_RXD_2 11 - GPIO19
PCI_AD11	0x144	PCI	0_0??0_0000 ⁶	00 - PCI_AD11 01 - VIU_DATA5 10 - FEC_RXD_1 11 - GPIO20
PCI_AD10	0x148	PCI	0_0??0_0000 ⁶	00 - PCI_AD10 01 - Reserved 10 - FEC_RXD_0 11 - GPIO21
PCI_AD09	0x14C	PCI_ST	0_0??0_0000 ⁶	00 - PCI_AD09 01 - Reserved 10 - FEC_RX_CLK 11 - GPIO22
PCI_AD08	0x150	PCI_ST	0_0??0_0000 ⁶	00 - PCI_AD08 01 - Reserved 10 - FEC_TX_CLK 11 - GPIO23
PCI_AD07	0x154	PCI	0_0??0_0000 ⁶	00 - PCI_AD07 01 - VIU_DATA7 10 - FEC_RX_ER 11 - GPIO24
PCI_AD06	0x158	PCI	0_0??0_0000 ⁶	00 - PCI_AD06 01 - Reserved 10 - FEC_RX_DV 11 - GPIO25
PCI_AD05	0x15C	PCI	0_0??0_0000 ⁶	00 - PCI_AD05 01 - Reserved 10 - FEC_TX_EN 11 - GPIO26
PCI_AD04	0x160	PCI	0_0??0_0000 ⁶	00 - PCI_AD04 01 - VIU_PIX_CLK 10 - FEC_TX_ER 11 - GPIO27
PCI_AD03	0x164	PCI	0_0??0_0000 ⁶	00 - PCI_AD03 01 - VIU_DATA6 10 - FEC_CRS 11 - GPIO0

Table 22-16. Pad IO Control Register Table (Sheet 9 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PCI_AD02	0x168	PCI	0_0??0_0000 ⁶	00 - PCI_AD02 01 - VIU_DATA8 10 - FEC_MDC 11 - GPIO1
PCI_AD01	0x16C	PCI	0_0??0_0000 ⁶	00 - PCI_AD01 01 - VIU_DATA9 10 - FEC_MDIO 11 - GPIO2
PCI_AD00	0x170	PCI	0_0??0_0000 ⁶	00 - PCI_AD00 01 - Reserved 10 - FEC_COL 11 - GPIO3
PCI_C/ $\overline{\text{BE}}$ 0	0x174	PCI	0_0??0_0000 ⁶	00 - PCI_C/ $\overline{\text{BE}}$ 0 01 - USB0_DATA5 10 - DIU_LD03 11 - GPIO4
PCI_C/ $\overline{\text{BE}}$ 1	0x178	PCI	0_0??0_0000 ⁶	00 - PCI_C/ $\overline{\text{BE}}$ 1 01 - USB0_DATA4 10 - DIU_LD02 11 - GPIO5
PCI_C/ $\overline{\text{BE}}$ 2	0x17C	PCI	0_0??0_0000 ⁶	00 - PCI_C/ $\overline{\text{BE}}$ 2 01 - USB0_DATA3 10 - DIU_LD00 11 - GPIO6
PCI_C/ $\overline{\text{BE}}$ 3	0x180	PCI	0_0??0_0000 ⁶	00 - PCI_C/ $\overline{\text{BE}}$ 3 01 - USB0_DATA2 10 - DIU_LD01 11 - GPIO7
PCI_GNT2	0x184	PCI	0_0??0_0000 ⁶	00 - PCI_GRANT2 01 - Reserved 10 - DIU_LD21 11 - GPIO8
PCI_REQ2	0x188	PCI	0_0??0_0000 ⁶	00 - PCI_REQ2 01 - Reserved 10 - DIU_LD20 11 - GPIO9
PCI_GNT1	0x18C	PCI	0_0??0_0000 ⁶	00 - PCI_GRANT1 01 - Reserved 10 - DIU_LD19 11 - GPIO10
PCI_REQ1	0x190	PCI	0_0??0_0000 ⁶	00 - PCI_REQ1 01 - Reserved 10 - DIU_LD18 11 - GPIO11

Table 22-16. Pad IO Control Register Table (Sheet 10 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PCI_GNT0	0x194	PCI	0_0??0_0000 ⁶	00 - PCI_GRANT0 01 - Reserved 10 - DIU_LD12 11 - GPIO12
PCI_REQ0	0x198	PCI	0_0??0_0000 ⁶	00 - PCI_REQ0 01 - Reserved 10 - DIU_LD11 11 - GPIO13
PCI_INTA	0x19C	PCI	0_0??0_0000 ⁶	00 - PCI_INTA 01 - Reserved 10 - DIU_LD15 11 - GPIO14
PCI_CLK	0x1A0	PCI	0_0000_0000	00 - PCI_CLK 01 - Reserved 10 - DIU_LD14 11 - GPIO15
PCI_RST_OUT	0x1A4	PCI	0_0??0_0000 ⁶	00 - PCI_RST_OUT 01 - Reserved 10 - DIU_LD13 11 - GPIO16
PCI_FRAME	0x1A8	PCI	0_0??0_0000 ⁶	00 - PCI_FRAME 01 - Reserved 10 - DIU_LD10 11 - GPIO17
PCI_IDSEL	0x1AC	PCI	0_0??0_0000 ⁶	00 - PCI_IDSEL 01 - Reserved 10 - DIU_LD07 11 - GPIO18
PCI_DEVSEL	0x1B0	PCI	0_0??0_0000 ⁶	00 - PCI_DEVSEL 01 - Reserved 10 - DIU_LD06 11 - GPIO19
PCI_IRDY	0x1B4	PCI	0_0??0_0000 ⁶	00 - PCI_IRDY 01 - USB0_DATA7 10 - DIU_LD05 11 - GPIO20
PCI_TRDY	0x1B8	PCI	0_0??0_0000 ⁶	00 - PCI_TRDY 01 - USB0_DATA6 10 - DIU_LD04 11 - GPIO21
PCI_STOP	0x1BC	PCI	0_0??0_0000 ⁶	00 - PCI_STOP 01 - USB0_DATA1 10 - DIU_LD08 11 - GPIO22

Table 22-16. Pad IO Control Register Table (Sheet 11 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PCI_PAR	0x1C0	PCI	0_0??0_0000 ⁶	00 - PCI_PAR 01 - USB0_DATA0 10 - DIU_LD09 11 - GPIO23
PCI_PERR	0x1C4	PCI	0_0??0_0000 ⁶	00 - PCI_PERR 01 - USB0_STOP 10 - DIU_LD16 11 - GPIO24
PCI_SERR	0x1C8	PCI	0_0??0_0000 ⁶	00 - PCI_SERR 01 - USB0_NEXT 10 - DIU_LD17 11 - GPIO25
SPDIF_TXCLK	0x1CC	STD_ST	0_0000_0100	00 - SPDIF_TXCLK 01 - FEC_RX_DV 10 - DIU_CLK 11 - GPIO26
SPDIF_TX	0x1D0	STD	0_0000_0000	00 - SPDIF_TX 01 - FEC_TX_ER 10 - DIU_VSYNC 11 - GPIO27
SPDIF_RX	0x1D4	STD	0_0000_0000	00 - SPDIF_RX 01 - FEC_CRS 10 - DIU_HSYNC 11 - GPIO0
I2C0_SCL	0x1D8	STD_ST	0_0000_01?? ⁷	00 - I2C0_SCL 01 - Reserved 10 - Reserved 11 - GPIO7/GPT7
I2C0_SDA	0x1DC	STD_ST	0_0000_01?? ⁷	00 - I2C0_SDA 01 - Reserved 10 - Reserved 11 - GPIO1
I2C1_SCL	0x1E0	STD_ST	0_0000_01?? ⁷	00 - I2C1_SCL 01 - Reserved 10 - SPDIF_TX 11 - GPIO2
I2C1_SDA	0x1E4	STD_ST	0_0000_01?? ⁷	00 - I2C1_SDA 01 - Reserved 10 - SPDIF_RX 11 - GPIO3
I2C2_SCL	0x1E8	STD_ST	0_0000_01?? ⁷	00 - I2C2_SCL 01 - Reserved 10 - CAN4_TX 11 - GPIO4

Table 22-16. Pad IO Control Register Table (Sheet 12 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
I2C2_SDA	0x1EC	STD_ST	0_0000_01??	00 - I2C2_SDA 01 - Reserved 10 - CAN4_RX 11 - GPIO5
IRQ0	0x1F0	STD	0_0000_0000	00 - IRQ0 01 - Reserved 10 - CAN3_TX 11 - GPIO4/GPT4
IRQ1	0x1F4	STD_ST	0_0000_0000	00 - IRQ1 01 - SPDIF_TXCLK 10 - CAN3_RX 11 - GPIO5/GPT5
CAN1_TX	0x1F8	STD	0_0000_00?? ⁷	00 - CAN1_TX 01 - Reserved 10 - Reserved 11 - GPIO6
CAN2_TX	0x1FC	STD	0_0000_00?? ⁷	00 - CAN2_TX 01 - Reserved 10 - Reserved 11 - GPIO8
J1850_TX	0x200	STD	0_0000_00?? ⁷	00 - J1850_TX 01 - Reserved 10 - GPIO4 11 - CAN4_TX
J1850_RX	0x204	STD_PU	0_0001_10?? ⁷	00 - J1850_RX 01 - Reserved 10 - LPC_CS6 11 - CAN4_RX
PSC_MCLK_IN	0x208	STD_ST	0_0000_0100	00 - PSC_MCLK_IN 01 - Reserved 10 - DIU_DE 11 - GPIO6/GPT6
PSC0_0	0x20C	STD_ST	0_0000_0100	00 - PSC0_0 01 - FEC_COL 10 - USB0_DATA7 11 - GPIO8
PSC0_1	0x210	STD	0_0000_0000	00 - PSC0_1 01 - FEC_TX_EN 10 - USB0_DATA6 11 - GPIO9
PSC0_2	0x214	STD_ST	0_0000_0000	00 - PSC0_2 01 - FEC_TX_CLK 10 - USB0_DATA5 11 - GPIO10

Table 22-16. Pad IO Control Register Table (Sheet 13 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC0_3	0x218	STD	0_0000_0000	00 - PSC0_3 01 - FEC_TXD_0 10 - USB0_DATA4 11 - GPIO11
PSC0_4	0x21C	STD	0_0000_0000	00 - PSC0_4 01 - FEC_TXD_1 10 - USB0_DATA3 11 - GPIO0/GPT0
PSC1_0	0x220	STD_ST	0_0000_0100	00 - PSC1_0 01 - FEC_TXD_2 10 - USB0_DATA2 11 - GPIO12
PSC1_1	0x224	STD	0_0000_0000	00 - PSC1_1 01 - FEC_TXD_3 10 - USB0_DATA1 11 - GPIO13
PSC1_2	0x228	STD	0_0000_0000	00 - PSC1_2 01 - FEC_MDC 10 - USB0_DATA0 11 - GPIO14
PSC1_3	0x22C	STD	0_0000_0000	00 - PSC1_3 01 - FEC_RX_ER 10 - USB0_STOP 11 - GPIO15
PSC1_4	0x230	STD	0_0000_0000	00 - PSC1_4 01 - FEC_RXD_3 10 - USB0_NEXT 11 - GPIO1/GPT1
PSC2_0	0x234	STD_ST	0_0000_0100	00 - PSC2_0 01 - FEC_RXD_2 10 - USB0_CLK 11 - GPIO16
PSC2_1	0x238	STD	0_0000_0000	00 - PSC2_1 01 - FEC_RXD_1 10 - USB0_DIR 11 - GPIO17
PSC2_2	0x23C	STD	0_0000_0000	00 - PSC2_2 01 - FEC_RXD_0 10 - Reserved 11 - GPIO18
PSC2_3	0x240	STD	0_0000_0000	00 - PSC2_3 01 - FEC_MDIO 10 - Reserved 11 - GPIO19

Table 22-16. Pad IO Control Register Table (Sheet 14 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC2_4	0x244	STD_ST	0_0000_0000	00 - PSC2_4 01 - FEC_RX_CLK 10 - Reserved 11 - GPIO2/GPT2
PSC3_0	0x248	STD_ST	0_0000_0100	00 - PSC3_0 01 - USB1_DATA0 10 - Reserved 11 - GPIO20
PSC3_1	0x24C	STD	0_0000_0000	00 - PSC3_1 01 - USB1_DATA1 10 - Reserved 11 - GPIO21
PSC3_2	0x250	STD	0_0000_0000	00 - PSC3_2 01 - USB1_DATA2 10 - Reserved 11 - GPIO22
PSC3_3	0x254	STD	0_0000_0000	00 - PSC3_3 01 - USB1_DATA3 10 - Reserved 11 - GPIO23
PSC3_4	0x258	STD	0_0000_0100	00 - PSC3_4 01 - LPC_CS6 10 - VIU_PIX_CLK 11 - GPIO3/GPT3
PSC4_0	0x25C	STD_ST	0_0000_0100	00 - PSC4_0 01 - USB1_DATA4 10 - VIU_DATA0 11 - GPIO24
PSC4_1	0x260	STD	0_0000_0000	00 - PSC4_1 01 - USB1_DATA5 10 - VIU_DATA1 11 - GPIO25
PSC4_2	0x264	STD	0_0000_0000	00 - PSC4_2 01 - USB1_DATA6 10 - VIU_DATA2 11 - GPIO26
PSC4_3	0x268	STD	0_0000_0000	00 - PSC4_3 01 - USB1_DATA7 10 - VIU_DATA3 11 - GPIO27
PSC4_4	0x26C	STD_PU	0_0001_1000	00 - PSC4_4 01 - NFC_CE2 10 - VIU_DATA4 11 - GPIO4/GPT4

Table 22-16. Pad IO Control Register Table (Sheet 15 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC5_0	0x270	STD_ST	0_0000_0100	00 - PSC5_0 01 - USB1_CLK 10 - VIU_DATA5 11 - GPIO8
PSC5_1	0x274	STD	0_0000_0000	00 - PSC5_1 01 - USB1_NEXT 10 - VIU_DATA6 11 - GPIO9
PSC5_2	0x278	STD	0_0000_0000	00 - PSC5_2 01 - USB1_STOP 10 - VIU_DATA7 11 - GPIO10
PSC5_3	0x27C	STD	0_0000_0000	00 - PSC5_3 01 - USB1_DIR 10 - VIU_DATA8 11 - GPIO11
PSC5_4	0x280	STD_PU	0_0001_1000	00 - PSC5_4 01 - NFC_CE3 10 - VIU_DATA9 11 - GPIO5/GPT5
PSC6_0	0x284	STD_ST	0_0000_0100	00 - PSC6_0 01 - LPC_TSIZ1 10 - DIU_CLK 11 - GPIO12
PSC6_1	0x288	STD	0_0000_0000	00 - PSC6_1 01 - LPC_TSIZ2 10 - DIU_HSYNC 11 - GPIO13
PSC6_2	0x28C	STD	0_0000_0000	00 - PSC6_2 01 - Reserved 10 - Reserved 11 - GPIO14
PSC6_3	0x290	STD	0_0000_0000	00 - PSC6_3 01 - Reserved 10 - Reserved 11 - GPIO15
PSC6_4	0x294	STD	0_0000_0000	00 - PSC6_4 01 - LPC_TS 10 - DIU_VSYNC 11 - GPIO6/GPT6
PSC7_0	0x298	STD_PU_ST	0_0000_0100	00 - PSC7_0 01 - SDHC_CMD 10 - DIU_LD23 11 - GPIO16

Table 22-16. Pad IO Control Register Table (Sheet 16 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC7_1	0x29C	STD_PU	0_0000_0000	00 - PSC7_1 01 - SDHC_D0 10 - DIU_LD22 11 - GPIO17
PSC7_2	0x2A0	STD_PU	0_0000_0000	00 - PSC7_2 01 - SDHC_D1_IRQ 10 - DIU_LD17 11 - GPIO18
PSC7_3	0x2A4	STD_PU	0_0000_0000	00 - PSC7_3 01 - SDHC_D2 10 - DIU_LD16 11 - GPIO19
PSC7_4	0x2A8	STD_PU	0_0000_0000	00 - PSC7_4 01 - SDHC_D3_CD 10 - DIU_LD09 11 - GPIO7/GPT7
PSC8_0	0x2AC	STD_ST	0_0000_0000	00 - PSC8_0 01 - Reserved 10 - DIU_LD08 11 - GPIO20
PSC8_1	0x2B0	STD	0_0000_0000	00 - PSC8_1 01 - Reserved 10 - DIU_LD01 11 - GPIO21
PSC8_2	0x2B4	STD	0_0000_0000	00 - PSC8_2 01 - Reserved 10 - DIU_LD00 11 - GPIO22
PSC8_3	0x2B8	STD	0_0000_0000	00 - PSC8_3 01 - Reserved 10 - DIU_LD02 11 - GPIO23
PSC8_4	0x2BC	STD	0_0000_0000	00 - PSC8_4 01 - SDHC_CLK 10 - DIU_LD03 11 - GPIO0/GPT0
PSC9_0	0x2C0	STD_ST	0_0000_0000	00 - PSC9_0 01 - Reserved 10 - DIU_LD04 11 - GPIO24
PSC9_1	0x2C4	STD	0_0000_0000	00 - PSC9_1 01 - Reserved 10 - DIU_LD05 11 - GPIO25

Table 22-16. Pad IO Control Register Table (Sheet 17 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC9_2	0x2C8	STD	0_0000_0000	00 - PSC9_2 01 - Reserved 10 - DIU_LD06 11 - GPIO26
PSC9_3	0x2CC	STD	0_0000_0000	00 - PSC9_3 01 - Reserved 10 - DIU_LD07 11 - GPIO27
PSC9_4	0x2D0	STD	0_0000_0000	00 - PSC9_4 01 - Reserved 10 - DIU_LD10 11 - GPIO1/GPT1
PSC10_0	0x2D4	STD_ST	0_0000_0000	00 - PSC10_0 01 - Reserved 10 - DIU_LD11 11 - GPIO8
PSC10_1	0x2D8	STD	0_0000_0000	00 - PSC10_1 01 - Reserved 10 - DIU_LD12 11 - GPIO9
PSC10_2	0x2DC	STD	0_0000_0000	00 - PSC10_2 01 - Reserved 10 - DIU_LD13 11 - GPIO10
PSC10_3	0x2E0	STD	0_0000_0000	00 - PSC10_3 01 - Reserved 10 - DIU_LD14 11 - GPIO11
PSC10_4	0x2E4	STD	0_0000_0000	00 - PSC10_4 01 - Reserved 10 - DIU_LD15 11 - GPIO2/GPT2
PSC11_0	0x2E8	STD_ST	0_0000_0000	00 - PSC11_0 01 - Reserved 10 - DIU_LD18 11 - GPIO12
PSC11_1	0x2EC	STD	0_0000_0000	00 - PSC11_1 01 - Reserved 10 - DIU_LD19 11 - GPIO13
PSC11_2	0x2F0	STD	0_0000_0000	00 - PSC11_2 01 - Reserved 10 - DIU_LD20 11 - GPIO14

Table 22-16. Pad IO Control Register Table (Sheet 18 of 18)

PAD Name	Address	Type	Reset Value [8:0]	FUNCMUX
PSC11_3	0x2F4	STD	0_0000_0000	00 - PSC11_3 01 - Reserved 10 - DIU_LD21 11 - GPIO15
PSC11_4	0x2F8	STD	0_0000_0000	00 - PSC11_4 01 - USBPHYDBG_IDDIGReserved 10 - DIU_DE 11 - GPIO3/GPT3
CKSTP_OUT	0x304	STD	0_0000_0000	00 - CKSTP_OUT 01 - TPA 10 - Reserved 11 - Reserved
USB_PHY_DRVVBUS	0x310	STD	0_0000_0000	00 - USB2_DRVVBUS 01 - Reserved 10 - Reserved 11 - Reserved

- Note:** 1) Slew Rate is 11 if LPC is the boot source. Otherwise it is 00.
2) Slew Rate is 11 if NFC is the boot source. Otherwise it is 00.
3) Slew Rate is 11 if LPC or 16-bit mode NFC is the boot source. Otherwise it is 00.
4) Slew Rate is 11 if LPC is the boot source and the extended LPC address bus (LPC AX [08:04]) is muxed with PATA control signals. Otherwise it is 00.
5) The LPC AX [09:04] signals are muxed over the NFC control lines in case the LPC_AX reset configuration is set to 10.
6) The PCI delay is 11 in case M66en reset configuration is 0. If the M66en reset configuration is 1, the PCI delay is 01.
7) Slew Rate is 11 if LPC is the boot source and the extended LPC address bus (LPC AX [09:04]) is muxed with NFC control signals. Otherwise it is 00.

Chapter 23

LocalPlus Bus (LPC)

23.1 Introduction

The LocalPlus Bus (LPC) is the external bus interface of the MPC5121e. This multi-function bus system supports interfacing to external boot ROM or flash memories, external SRAM memories, or other memory mapped devices. See [Figure 23-1](#) for a block diagram of the LPC.

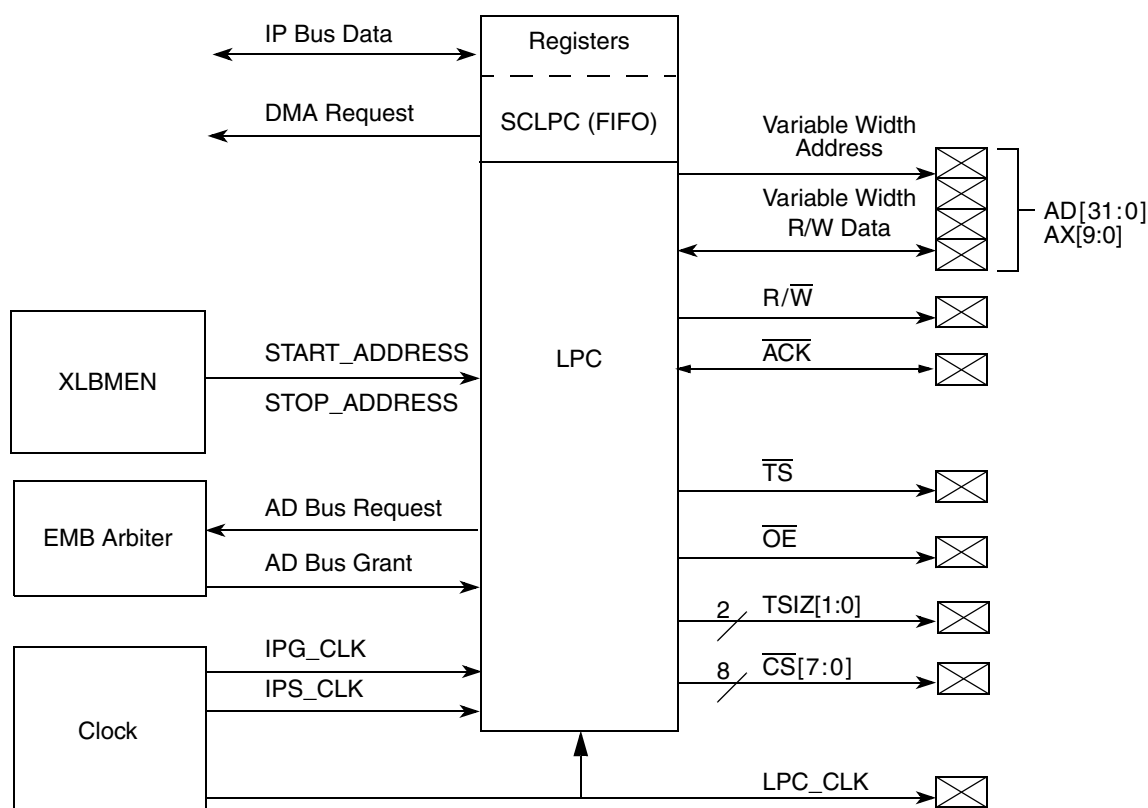


Figure 23-1. LPC Block Diagram

NOTE

AD is the shared address/data bus. AX is the address extension bus.

23.1.1 Features

The LPC includes:

- Interface to memory mapped or chip selected devices
- Two main modes of operation:
 - Non-muxed modes
 - Address up to 32 bits, data 8, 16, or 32 bits
 - Muxed modes (address latch enable (ALE) usage)
 - Address up to 32 bits, data 8, 16, or 32 bits
 - Programmable ALE level
- 8 chip-select (CS) signals
 - Programmable wait states per CS
 - Programmable deadcycles per CS
 - Programmable holdcycles per CS
 - Programmable byte swapping per CS
- Configurable boot interface supporting Power Architecture architecture code execution
- Dynamic bus sizing
- Support of burst mode flash devices (up to 32 byte bursts)
 - Synchronous burst read
 - Synchronous burst write
 - Asynchronous burst read (page mode)
 - Asynchronous burst write (page mode)
- DMA support allows data movement independently from the CPU (up to 56 byte bursts)
- No support of misaligned accesses. This also includes transfers of 3, 5, 6, and 7 bytes.

Table 23-1. Signal Properties

Name	Function	I/O	Reset	Pullup
AD[31:0]	Address and data lines	I/O	0	—
AX[31:0]	Non-muxed mode: address lines Muxed mode: address latch enable (AX[0]) Transfer size (AX[2:1]) Transfer start (AX[3])	O	0	—
$\overline{\text{ACK}}$	No burst transaction: Acknowledge can shorten a transaction Burst transaction: Indicates that a burst transaction is ongoing.	I/O	—	Pullup
LPC_CLK	LPC clock	O	—	—
$\overline{\text{CS}}[7:0]$	Chip select	O	1	—
$\overline{\text{OE}}$	Output enable	O	1	—
$\text{R}/\overline{\text{W}}$	Read/write bar	O	1	—
TSIZ[2:0]	Transfer size	O	0	—
$\overline{\text{TS}}$	Transfer start	O	0	—

23.2 Memory Map and Register Definition

Table 23-2. LPC Block Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x000	Chip Select 0/Boot Configuration Register	R/W	23.2.1.1.1/23-4
0x004	Chip Select 1 Configuration Register	R/W	23.2.1.1.2/23-8
0x008	Chip Select 2 Configuration Register	R/W	23.2.1.1.2/23-8
0x00C	Chip Select 3 Configuration Register	R/W	23.2.1.1.2/23-8
0x010	Chip Select 4 Configuration Register	R/W	23.2.1.1.2/23-8
0x014	Chip Select 5 Configuration Register	R/W	23.2.1.1.2/23-8
0x018	Chip Select 6 Configuration Register	R/W	23.2.1.1.2/23-8
0x01C	Chip Select 7 Configuration Register	R/W	23.2.1.1.2/23-8
0x020	Chip Select Control Register	R/W	23.2.1.1.3/23-12
0x024	Chip Select Status Register	R/W	23.2.1.1.4/23-13
0x028	Chip Select Burst Control	R/W	23.2.1.1.5/23-14
0x02C	Chip Select Deadcycle Control Register	R/W	23.2.1.1.6/23-15
0x030	Chip Select Holdcycle Control Register	R/W	23.2.1.1.7/23-16
0x034	Address Latch Timing Register	R/W	23.2.1.1.8/23-17
0x100	SCLPC Packet Size Register	R/W	23.2.1.2.1/23-18
0x104	SCLPC Start Address Register	R/W	23.2.1.2.2/23-19
0x108	SCLPC Control Register	R/W	23.2.1.2.3/23-20
0x10C	SCLPC Enable Register	R/W	23.2.1.2.4/23-21
0x110	SCLPC NextAddress Register	R/W	23.2.1.2.5/23-22
0x114	SCLPC Status Register	R/W	23.2.1.2.5/23-22
0x118	SCLPC Bytes Done Register	R/W	23.2.1.2.6/23-23
0x11C	EMB Share Counter Register	R/W	23.2.1.2.7/23-24
0x120	EMB Pause Control Register	R/W	23.2.1.2.8/23-25
0x140	LPC RX/TX FIFO Data Word Register	R/W	23.2.1.3.1/23-26
0x144	LPC RX/TX FIFO Status Register	R/W	23.2.1.3.2/23-27
0x148	LPC RX/TX FIFO Control Register	R/W	23.2.1.3.3/23-28
0x14C	LPC RX/TX FIFO Alarm Register	R/W	23.2.1.3.4/23-29

23.2.1 Register Descriptions

23.2.1.1 Chip Select/LPC Registers—0x0000

There are fourteen 32-bit chip select/LPC (CS/LP) registers. These registers are located at an offset from IMMR of <lpc_addr_offset>. Register addresses are relative to this offset. Therefore, the actual register address is IMMR + <lpc_addr_offset> + register address.

The following registers are available:

Section 23.2.1.1.1, “Chip Select 0/Boot Configuration Register” (0x0000)

Section 23.2.1.1.2, “Chip Select[1:7] Configuration Registers” (0x0004 - 0x0001C)

Section 23.2.1.1.3, “Chip Select Control Register” (0x0020)

Section 23.2.1.1.4, “Chip Select Status Register” (0x0024)

Section 23.2.1.1.5, “Chip Select Burst Control Register” (0x0028)

Section 23.2.1.1.6, “Chip Select Deadcycle Control Register” (0x002C)

Section 23.2.1.1.7, “Chip Select Holdcycle Control Register” (0x0030)

Section 23.2.1.1.8, “Address Latch Timing Register” (0x0034)

23.2.1.1.1 Chip Select 0/Boot Configuration Register

Offset 0x0000 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	cfg	0	0	1	0	0	cfg	cfg	0	cfg	0	0	0	0	0	1

Figure 23-2. Chip Select 0/Boot Configuration Register
(Register is repeated for reference.)

Table 23-3. Chip Select 0/Boot Configuration Register Field Descriptions (Sheet 1 of 4)

Field	Description
WaitP	<p>Number of wait states to insert. Can be applied as a prescale to WaitX or used by itself, as dictated by the WTyp bits (see below). Wait states control the number of LPC clocks for which the corresponding CS pin remains active in a non-burstable transaction. The default wait time is two LPC clocks. For example, if the WaitP is set to four, the CS is asserted as maximum for six clocks. Acknowledge can shorten the WaitP time, but not the fixed two LPC clock time.</p> <p>This parameter describes the time before the burst signal is asserted for a burst transaction. An additional two clocks are also available for this read operation. There are an additional two and one-half clocks available for write operations.</p>

Offset 0x0000Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	cfg	0	0	1	0	0	cfg	cfg	0	cfg	0	0	0	0	0	1

Figure 23-2. Chip Select 0/Boot Configuration Register
(Register is repeated for reference.)

Table 23-3. Chip Select 0/Boot Configuration Register Field Descriptions (Sheet 2 of 4)

Field	Description
WaitX	The base number of wait states to insert, or to be combined with WaitP, as dictated by the WaitP bits (see below). See the WaitP description.
MX	The MX bit specifies whether a transaction operates a muxed or non-muxed. A muxed transaction presents address and data in different tenures. ALE is asserted during the address tenure. At the end of ALE, the address remains driven for at least one LPC clock before the \overline{CSx} pin is asserted. 0 Non-muxed 1 Muxed
ALEV	ALE level. 0 ALE is active low 1 ALE is active high
AA	ACK active. This bit defines whether \overline{ACK} input is active or not. 1 Programmed wait states can be overridden if/when the external device drives the \overline{ACK} input low. Wait states remain in effect. If no \overline{ACK} is received, the cycle terminates at the end of the wait state period. 0 \overline{ACK} input is not active and cannot shorten the wait state time.
CE	An individual enable bit that allows CS operation for the corresponding CS pin. CE must be high to allow operation. The chip select control register ME bit must also be high, except when CS[0] is used for boot ROM. 1 External CS is enabled 0 External CS is disabled
ALEN	ALE length 00 ALE width is one LPC clock 01 ALE width is two LPC clocks 10 ALE width is three LPC clocks 11 ALE width is four LPC clocks Note: ALE length configures not only the width of the ALE assertion, but also the width of the isolation cycle between ALE deassertion and CS assertion.
DS	Data size field, which represents the device data bus size (in bytes): 00 1 byte 01 2 bytes 11 4 bytes Note: Table 23-25 and Table 23-29 show on which AD lines the data is located.

Offset 0x0000Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	cfg	0	0	1	0	0	cfg	cfg	0	cfg	0	0	0	0	0	1

Figure 23-2. Chip Select 0/Boot Configuration Register
(Register is repeated for reference.)

Table 23-3. Chip Select 0/Boot Configuration Register Field Descriptions (Sheet 3 of 4)

Field	Description
BM	Burst mode 0 Synchronous burst mode 1 Asynchronous burst mode (page mode) Note: The asynchronous burst mode setting is only valid for non-muxed transactions. If this bit is set and muxed mode is enabled, a synchronous burst is performed. Note: This bit has no influence if bursting (read or write) is not enabled.
ADDRM	Address mode 0 Byte addressing 1 Short or word addressing An 8-bit data bus always uses a byte addressing. Note: Table 23-25 and Table 23-29 show on which AD/AX lines the address is located.
WTyp	Wait state type bits that define the application of wait states contained in WaitP and WaitX fields as follows: 00 WaitX is applied to read and write cycles (WaitP is ignored) 01 WaitX is applied to read cycles; WaitP is applied to write cycles 10 WaitX is applied to reads; WaitP/WaitX (16-bit value) is applied to writes 11 WaitP/WaitX (as a full 16-bit value) is applied to reads and writes
WS	Write swap bit. If high, endian byte swapping occurs during writes to a device. <ul style="list-style-type: none"> For 8-bit devices, this bit has no effect For 16-bit devices, byte swapping can occur For 32-bit devices (possible in muxed mode only), byte swapping can occur 1 Swapping can occur 0 Swapping cannot occur A 2-byte swap is AB to BA; a 4-byte swap is ABCD to DCBA. Note: Transactions at less than the defined port size (i.e., data size) apply swapping rules as above, according to the current transaction size.
RS	Read swap bit. Same as ws, but swapping is done when reading data from a device. <ul style="list-style-type: none"> Swapping can occur Swapping cannot occur Note: Transactions at less than the defined port size (i.e., data size) apply swapping rules as above, according to the current transaction size.

Offset 0x0000Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	cfg	0	0	1	0	0	cfg	cfg	0	cfg	0	0	0	0	0	1

Figure 23-2. Chip Select 0/Boot Configuration Register
(Register is repeated for reference.)

Table 23-3. Chip Select 0/Boot Configuration Register Field Descriptions (Sheet 4 of 4)

Field	Description
WO	Write-only bit. If the bit is high, the device is treated as a write-only device. An attempted read access can result in an interrupt (as dictated by the Chip Select Control Register ie bit). In any case, no transaction is presented to the device.
RO	Read-only bit. If the bit is high, the device is treated as a read-only device. An attempted write access can result in an interrupt (as dictated by the Chip Select Control Register ie bit). In any case, no transaction is presented to the device. Note: This bit is high from reset, indicating the boot device is read-only.

23.2.1.1.2 Chip Select[1:7] Configuration Registers

Offset: Chip Select 1 Configuration Register 0x0004 Access: User read/write

Chip Select 2 Configuration Register 0x0008

Chip Select 3 Configuration Register 0x000C

Chip Select 4 Configuration Register 0x0010

Chip Select 5 Configuration Register 0x0014

Chip Select 6 Configuration Register 0x0018

Chip Select 7 Configuration Register 0x001C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-3. Chip Select[1:7] Configuration Registers
(Register is repeated for reference.)

Table 23-4. Chip Select[1:7] Configuration Registers Field Descriptions (Sheet 1 of 4)

Field	Description
WaitP	Number of wait states to insert. Can be applied as a prescale to WaitX or used by itself, as dictated by the WTyp bits (see below). Wait states control the number of LPC clocks for which the corresponding CS pin remains active in a non-burstable transaction. The default wait time is two LPC clocks. For example, if the WaitP is set to four, the CS is asserted as maximum for six clocks. Acknowledgment can shorten the WaitP time, but not the fixed two LPC clock time. This parameter describes the time before the burst signal is asserted for a burst transaction. An additional two clocks are also available for this read operation. There are an additional two and one-half clocks available for write operations.
WaitX	The base number of wait states to insert, or to be combined with WaitP, as dictated by the WTyp bits (see below). See the WaitP description.
MX	The MX bit specifies whether a transaction operates as muxed or non-muxed. A muxed transaction presents address and data in different tenures. ALE is asserted during the address tenure. At the end of ALE, the address remains driven for at least one LPC clock before the \overline{CSx} pin is asserted. 0 Non-muxed 1 Muxed
ALEV	ALE level 0 ALE is active low 1 ALE is active high
AA	ACK active. This bit defines whether \overline{ACK} input is active or not. 1 Programmed wait states can be overridden if/when the external device drives the \overline{ACK} input low. Wait states remains in effect. If no \overline{ACK} is received, the cycle terminates at the end of wait state period. 0 \overline{ACK} input is not active and cannot shorten the wait state time.

Offset: Chip Select 1 Configuration Register 0x0004 Access: User read/write

Chip Select 2 Configuration Register 0x0008

Chip Select 3 Configuration Register 0x000C

Chip Select 4 Configuration Register 0x0010

Chip Select 5 Configuration Register 0x0014

Chip Select 6 Configuration Register 0x0018

Chip Select 7 Configuration Register 0x001C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN	DS		BM	ADD RM	WTyp		WS	RS	WO	RO	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-3. Chip Select[1:7] Configuration Registers
(Register is repeated for reference.)

Table 23-4. Chip Select[1:7] Configuration Registers Field Descriptions (Sheet 2 of 4)

Field	Description
CE	An individual enable bit that allows CS operation for the corresponding CS pin. CE must be high to allow operation. The chip select control register ME bit must also be high, except when CS[0] is used for boot ROM. 1 External CS is enabled 0 External CS is disabled
ALEN	ALE length 00 ALE width is one LPC clock 01 ALE width is two LPC clocks 10 ALE width is three LPC clocks 11 ALE width is four LPC clocks Note: ALE length configures not only the width of the ALE assertion, but also the width of the isolation cycle between ALE deassertion and CS assertion.
DS	Data size field that represents the device data bus size (in bytes): 00 1 byte 01 2 bytes 11 4 bytes Note: Table 23-25 and Table 23-29 show on which AD lines the data is located.
BM	Burst mode 0 Synchronous burst mode 1 Asynchronous burst mode (page mode) Note: The asynchronous burst mode setting is only valid for non-muxed transactions. If this bit is set and muxed mode is enabled, a synchronous burst is performed. Note: This bit has no influence if bursting (read or write) is not enabled.

Offset: Chip Select 1 Configuration Register 0x0004 Access: User read/write

Chip Select 2 Configuration Register 0x0008

Chip Select 3 Configuration Register 0x000C

Chip Select 4 Configuration Register 0x0010

Chip Select 5 Configuration Register 0x0014

Chip Select 6 Configuration Register 0x0018

Chip Select 7 Configuration Register 0x001C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-3. Chip Select[1:7] Configuration Registers
(Register is repeated for reference.)

Table 23-4. Chip Select[1:7] Configuration Registers Field Descriptions (Sheet 3 of 4)

Field	Description
ADDRM	Address mode 0 Byte addressing 1 Short or word addressing An 8-bit data bus always uses a byte addressing. Note: Table 23-25 and Table 23-29 show on which AD/AX lines the address is located.
WTyp	Wait state type bits that define the application of wait states contained in the WaitP and WaitX fields, as follows: 00 WaitX is applied to read and write cycles (WaitP is ignored) 01 WaitX is applied to read cycles; WaitP is applied to write cycles 10 WaitX is applied to reads; WaitP/WaitX (16-bit value) is applied to writes 11 WaitP/WaitX (as a full 16-bit value) is applied to reads and writes
WS	Write swap bit. If high, endian byte swapping occurs during writes to a device. • For 8-bit devices, this bit has no effect • For 16-bit devices, byte swapping can occur • For 32-bit devices (possible in muxed mode only), byte swapping can occur 1 Swapping can occur 0 Swapping cannot occur A 2-byte swap is AB to BA; a 4-byte swap is ABCD to DCBA. Note: Transactions at less than the defined port size (i.e., data size) apply swapping rules as above, according to the current transaction size.
RS	Read swap bit. Same as WS, but swapping is done when reading data from a device. 1 Swapping can occur 0 Swapping cannot occur Note: Transactions at less than the defined port size (i.e., data size) apply swapping rules as above, according to the current transaction size.

Offset: Chip Select 1 Configuration Register 0x0004 Access: User read/write

Chip Select 2 Configuration Register 0x0008

Chip Select 3 Configuration Register 0x000C

Chip Select 4 Configuration Register 0x0010

Chip Select 5 Configuration Register 0x0014

Chip Select 6 Configuration Register 0x0018

Chip Select 7 Configuration Register 0x001C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WaitP								WaitX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MX	ALEV	AA	CE	ALEN		DS		BM	ADD RM	WTyp		WS	RS	WO	RO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-3. Chip Select[1:7] Configuration Registers
(Register is repeated for reference.)

Table 23-4. Chip Select[1:7] Configuration Registers Field Descriptions (Sheet 4 of 4)

Field	Description
WO	Write-only bit. If the bit is high, the device is treated as a write-only device. An attempted read access can results in an interrupt (as dictated by the Chip Select Control Register ie bit). In any case, no transaction is presented to the device.
RO	Read only bit. If the bit is high, the device is treated as a read-only device. An attempted write access can results in an interrupt (as dictated by the Chip Select Control Register IE bit). In any case, no transaction is presented to the device.

23.2.1.1.3 Chip Select Control Register

Offset 0x0020Access: User read/write

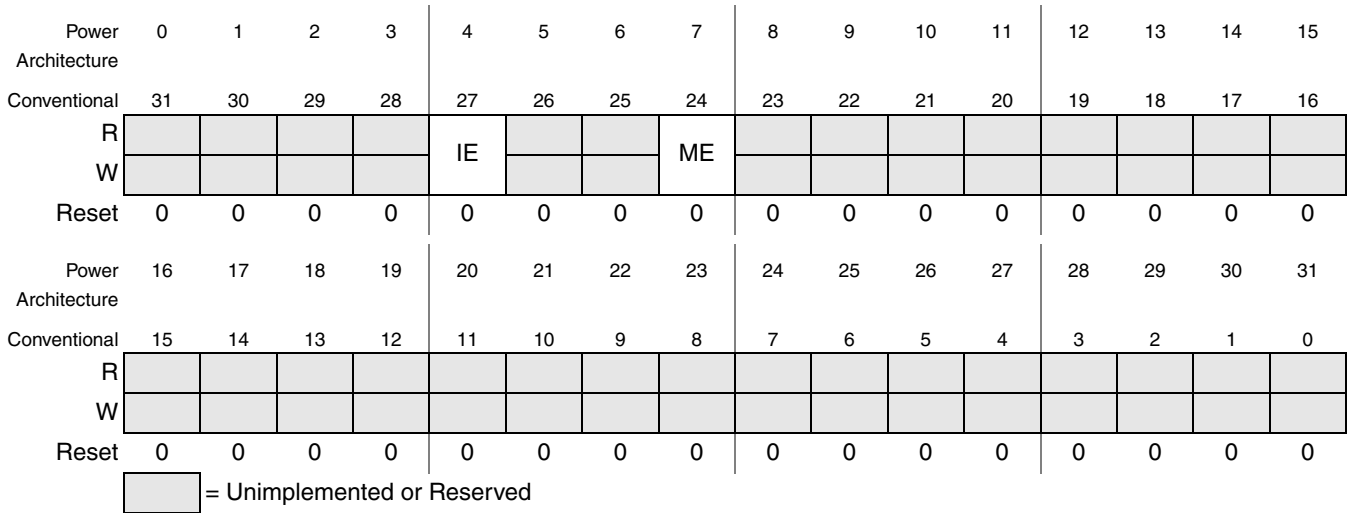


Figure 23-4. Chip Select Control Register

Table 23-5. Chip Select Control Register Field Descriptions

Field	Description
IE	Interrupt enable bit. An interrupt can be generated if a write is initiated to a read-only define CS; a read is initiated to a write-only defined CS; an SCLPC state machine finishes transfer (normal or abort); or an SCLPC FIFO detects FIFO errors (underrun or overrun).
ME	Master enable bit that is a global module enable bit. If this bit is low, register access can continue to occur, but no external transactions are accepted. However, ME does not affect boot ROM operation on $\overline{CS}[0]$. For software to disable $\overline{CS}[0]$, it must write 0 to the chip select boot ROM configuration register enable bit (CE).

23.2.1.1.4 Chip Select Status Register

Offset 0x0024 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W			WOerr	ROerr												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 23-5. Chip Select Status Register

Table 23-6. Chip Select Status Register Field Descriptions

Field	Description
WOerr	Write-Only Error. If 1, it indicates a read access was attempted on a device marked as write-only. This is a sticky bit and must be written with 1 to be cleared. This status bit is always active, regardless of the bus error enable bit. The CS number that relates to the error is reflected in the CSxerr field. An interrupt is also generated if the IE bit is set.
ROerr	Read-Only Error. If 1, it indicates a write access was attempted on a device marked as read-only. This is a sticky bit and must be written with 1 to be cleared. This status bit is always active, regardless of the bus error enable bit. The CS number that relates to the error is reflected in the CSxerr field. An interrupt is also generated if the IE bit is set.
CSxerr	Chip select error that indicates CS number associated with last Write-Only or Read-Only Error, as long as only one error type happens (write- or read-only). When the other type is happening, CSxerr points to the first violating CS of the opposite type. This is the case until one of the Error flags gets cleared.

23.2.1.1.5 Chip Select Burst Control Register

Offset 0x0028 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CW7	SLB7	BWE7	BRE7	CW6	SLB6	BWE6	BRE6	CW5	SLB5	BWE5	BRE5	CW4	SLB4	BWE4	BRE4
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CW3	SLB3	BWE3	BRE3	CW2	SLB2	BWE2	BRE2	CW1	SLB1	BWE1	BRE1	CW0	SLB0	BWE0	BRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-6. Chip Select Burst Control Register
(Register is repeated for reference.)

Table 23-7. Chip Select Burst Control Field Descriptions

Field	Description
CW7	Chip select 7 cache wrap-capable; set if a device burst can perform PPC cache wrap. Note: Cache wrap means the external device supports a wrap-around mechanism at addresses 0x20, 0x40, 0x60,... and so on. For example, 0x8-0xC-0x10-0x14-0x18-0x1C-0x0-0x4.
SLB7	Chip select 7 short/long burst; set 0 for short burst only, 1 for long burst-capable. Short burst are limited to 8 bytes, used for instruction fetches. Long burst-capable means the device can do a 16, 24, 32, 40, 48, or 56-byte burst. The length of the burst depends on the amount of data which should be transferred at once.
BWE7	Chip select 7 burst write enable; set 1 to enable device bursting for a given chip select. This bit must be set to enable any bursting writes.
BRE7	Chip select 7 burst read enable, 1 to enable device bursting for given chip select. Must be set to enable bursting reads.
CW6 - CW0	Same as CW7, but for CS6 - CS0/Boot
SLB6 - SLB0	Same as SLB7, but for CS6 - CS0/Boot
BWE6 - BWE0	Same as BWE7, but for CS6 - CS0/Boot
BRE6 - BRE0	Same as BRE7, but for CS6 - CS0/Boot

23.2.1.1.6 Chip Select Deadcycle Control Register

Offset 0x002CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R			DC7				DC6				DC5				DC4	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			DC3				DC2				DC1				DC0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

= Unimplemented or Reserved

Figure 23-7. Chip Select Deadcycle Control Register

Table 23-8. Chip Select Deadcycle Control Register Field Descriptions

Field	Description
DC7	<p>Deadcycles can be specified as 0 to 3. Deadcycles are added to the end of a Chip Select 7 read access and occur in addition to any cycles that may already exist. These cycles provide the device additional time to tri-state its bus after a read operation.</p> <p>00 device can drive data one LPC clock cycle after CS deassertion</p> <p>01 device can drive data two LPC clock cycle after CS deassertion</p> <p>10 device can drive data three LPC clock cycle after CS deassertion</p> <p>11 device can drive data four LPC clock cycle after CS deassertion</p>
DC6 - DC0	Same as DC7, but for CS6 - CS0/Boot

23.2.1.1.7 Chip Select Holdcycle Control Register

Offset 0x0030 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R			HC7				HC6				HC5				HC4	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			HC3				HC2				HC1				HC0	
W																
Reset	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

= Unimplemented or Reserved

Figure 23-8. Chip Select Holdcycle Control Register

Table 23-9. Chip Select Holdcycle Control Register Field Descriptions

Field	Description
HC7	<p>Holdcycles can be specified as 0 to 3. Holdcycles are added to the end of a Chip Select 7 write access and occur in addition to any cycles that may already exist. These cycles provide the device additional time to latch the data from the bus after a write operation.</p> <p>00 Data is valid one LPC clock cycle after CS deassertion</p> <p>01 Data is valid two LPC clock cycle after CS deassertion</p> <p>10 Data is valid three LPC clock cycle after CS deassertion</p> <p>11 Data is valid four LPC clock cycle after CS deassertion</p> <p>Note: For a write burst transaction the data is valid a half cycle less.</p>
HC6-HC0	Same as HC7, but for CS6 - CS0/Boot

23.2.1.1.8 Address Latch Timing Register

Offset 0x0034 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	ALT7	ALT6	ALT5	ALT4	ALT3	ALT2	ALT1	ALT0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 23-9. Address Latch Timing Control Register

Table 23-10. Address Latch Timing Register Field Descriptions

Field	Description
ALT7	Chip Select 7 address latch timing modification for multiplexed mode. 0 CS is asserted together with the assertion of Address latch (ALE). 1 CS is asserted (ALEN + 1) x LPC_CLK clocks after the deassertion of ALE.
ALT6-ALT0	Same as ALT7, but for CS6 - CS0/Boot

23.2.1.2 SCLPC Registers—0x0100

There are seven 32-bit registers for the LPC (SCLPC). These registers are located at an offset from IMMR of $\langle \text{lpc_addr_offset} \rangle + 0x0100$. Register addresses are relative to this offset. Therefore, the actual register address is $\text{IMMR} + \langle \text{lpc_addr_offset} \rangle + \text{register address}$.

The following registers are available:

Section 23.2.1.2.1, “SCLPC Packet Size Register” (0x0100)

Section 23.2.1.2.3, “SCLPC Control Register” (0x0108)

Section 23.2.1.2.4, “SCLPC Enable Register” (0x010C)

Section 23.2.1.2.5, “SCLPC Status Register” (0x0114)

Section 23.2.1.2.6, “SCLPC Bytes Done Register” (0x0118)

Section 23.2.1.2.7, “EMB Share and Wait Count Register” (0x011C)

Section 23.2.1.2.8, “EMB Pause Control Register” (0x0120)

23.2.1.2.1 SCLPC Packet Size Register

Offset 0x0100 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	Packet Size														
W	Restart															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Packet Size															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-10. SCLPC Packet Size Register

Table 23-11. SCLPC Packet Size Register Field Descriptions

Field	Description
Restart	Writing a 1 to this bit begins a SCLPC transfer. It clears automatically and always reads back as 0. Note: Start transfers after LPC FIFO and SCLPC are configured.
Packet Size	This 31-bit field represents the number of bytes SCLPC must transact before going idle and waiting for a restart. Note: The co-location of the restart bit and the packet_size field allows software to restart a transaction and change the packet_size in a single write. Maximum packet size is 2G-1 bytes. Note: Packet Size need to be either a multiple of BPT setting (SCLPC Enable Register Field Descriptions) or of 8.

23.2.1.2.2 SCLPC Start Address Register

Offset 0x0104 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Start Address															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Start Address															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-11. SCLPC Start Address Register

Table 23-12. SCLPC Start Address Register Field Descriptions

Field	Description
Start Address	The address of the first byte in the packet to be sent. Note: Start Address need to be either a multiple of BPT setting (SCLPC Enable Register Field Descriptions) or of 8.

23.2.1.2.3 SCLPC Control Register

Offset 0x0108 Access: User read/write

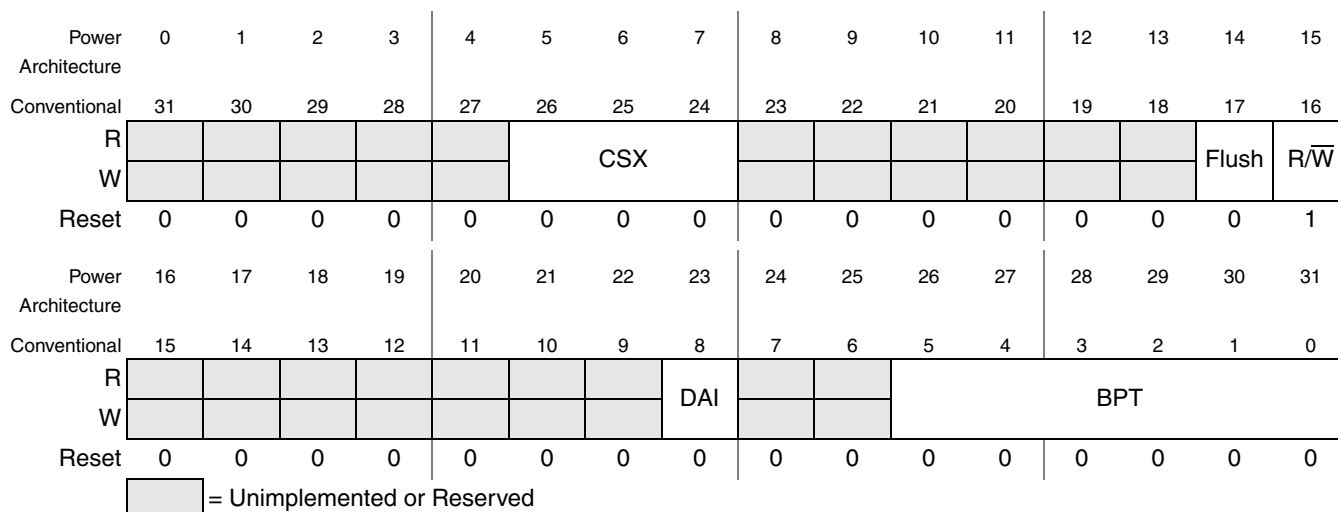


Figure 23-12. SCLPC Control Register

Table 23-13. SCLPC Control Register Field Descriptions

Field	Description
CSX	This field should be written with the chip select number associated with each SCLPC transaction. Note: LPC configuration registers associated with this CS also affect SCLPC transactions. The two work together.
Flush	If set to 1, this bit enables the assertion of the DMA request at the completion of a read packet, regardless of the actual state of the physical FIFO alarm. The DMA request deasserts after the FIFO empties.
R/W	Read/Write bar that controls the direction of the SCLPC transaction. 1 SCLPC reads from the device; i.e., FIFO receive 0 SCLPC writes to the device; i.e., FIFO transmit
DAI	Disable auto increment. Normally, SCLPC and LPC present sequential incrementing addresses to the device as the packet proceeds. If the device is operating as a single address FIFO, the DAI bit should be set to 1. When set, addresses to the device are stuck at start_address for every transaction. For DAI operation, the BPT field must be set to the port size of the device.
BPT	Bytes per transaction that indicates the number of bytes per transaction. The only valid entries in this field are 1, 2, 4, 8, 16, 24, 32, 40, 48 and 56. Start_address and packet_size values must be aligned/multiples of BPT or multiples of 8. BPT should be set to the device port size for DAI operation.

23.2.1.2.4 SCLPC Enable Register

Offset 0x010C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R								RC								RF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R							AIE	NIE								ME
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 23-13. SCLPC Enable Register

Table 23-14. SCLPC Enable Register Field Descriptions

Field	Description
RC	Reset controller. This bit allows for a software reset of the SCLPC state machine. Writing a 1 to this bit resets the SCLPC state machine. Reset is maintained as long as this bit is high. Software must write this bit low to release the reset and start operation. Note: Although RC does not reset this register interface, it does clear interrupt and interrupt status conditions. Never reset the SCLPC Controller during a transaction (TX or RX).
RF	Reset FIFO. This is the FIFO software reset bit. Writing a 1 to this bit resets the SCLPC FIFO. For normal operation, the FIFO must not be in reset. Resetting the FIFO clears the FIFO of data and resets its read/write pointers and the status bits, but it does not disturb previously programmed alarm and granularity settings. Note: It's recommended that software set and clear the RC and RF bits prior to programming and starting a packet.
AIE	Abort interrupt enable. If set, and a FIFO error occurs during packet transmission, a CPU interrupt from SCLPC is generated. In any case, the packet is terminated and an abort status bit is set.
NIE	Normal interrupt enable. If set, this bit enables a CPU interrupt to occur at the end of a normally terminated packet. There is also an NT status bit that sets in any case.
ME	Master enable. This bit must be set to 1 to generate a restart to the SCLPC state machine. Restart is achieved by writing 1 to byte 0 of the packet_size register. This ME bit must also be set for a restart to occur. Note: If ME is low (inactive), it also clears interrupt and interrupt status.

23.2.1.2.6 SCLPC Bytes Done Register

Offset 0x0118 Access: User read/write

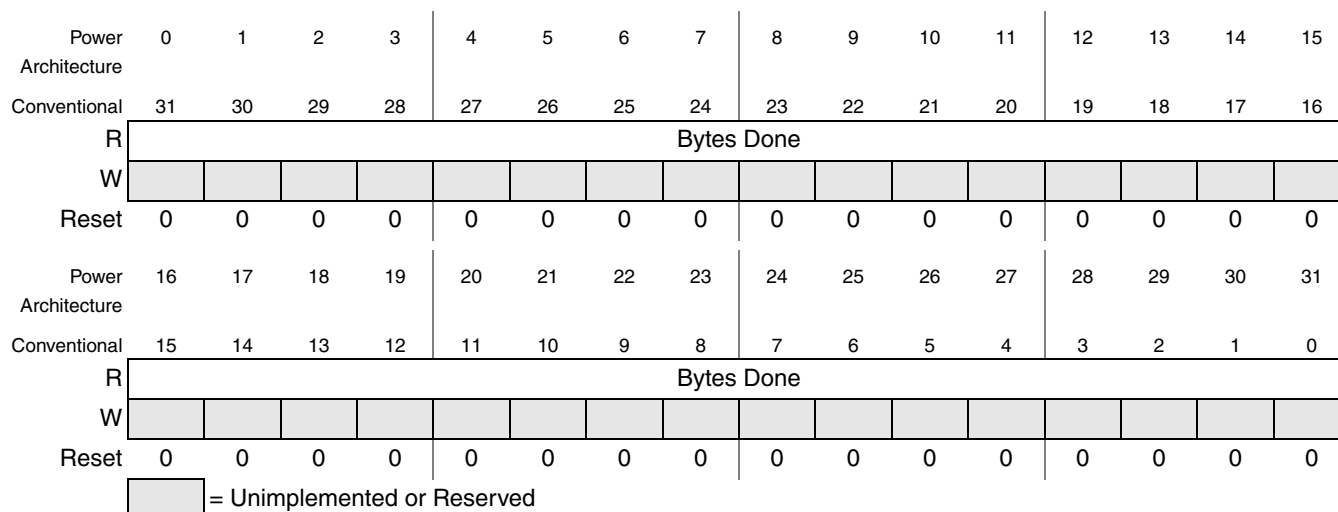


Figure 23-15. SCLPC Bytes Done Register

Table 23-16. SCLPC Bytes Done Register Field Descriptions

Field	Description
Bytes Done	Bytes done is updated dynamically by the SCLPC state machine to represent the actual number of bytes transmitted at a given point in time. At the normal conclusion of a packet, the BYTES_DONE field should match the PACKET_SIZE field.

23.2.1.2.7 EMB Share and Wait Count Register

This register configures the EMB arbiter and belong to the LPC functionality.

Offset 0x011C Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	emb_share_count															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	nfc_wait_count					ata_wait_count					lpc_wait_count				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-16. EMB Share and Wait Count Register

Table 23-17. EMB Share and Wait Count Register Field Descriptions

Field	Description
emb_share_count	This 16-bit value controls the length of the time slot assigned to ATA or NFC transactions before an SCLPC or CSB LPC (if the lpc_p bit is set in the emb pause control register) request starts to pause the other modules.
nfc_wait_count	This 5-bit value controls how long the bus remains assigned to NFC after the bus goes idle.
ata_wait_count	This 5-bit value controls how long the bus remains assigned to ATA after the bus goes idle.
lpc_wait_count	This 5-bit value controls how long the bus remains assigned to LPC after the bus goes idle.

23.2.1.2.8 EMB Pause Control Register

This register configures the EMB arbiter and belong to the LPC functionality.

Offset 0x08 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R															ATA_P	LPC_P
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 23-17. EMB Pause Control Register
(Register is repeated for reference.)

Table 23-18. EMB Pause Control Register Field Descriptions

Field	Description
ATA_P	ATA PIO pause disable 0 ATA PIO request can pause an ongoing SCLPC transaction. 1 ATA PIO request cannot pause an ongoing SCLPC transaction. The SCLPC transaction is allowed to finish the transfer before EMB deasserts the LPC bus grant.
LPC_P	LPC CSB pause disable 0 LPC CSB request immediately pauses an ATA DMA and NFC transfer. 1 LPC CSB request does not immediately pause an ATA DMA and NFC transfer. The pause is initiated after the emb_share_count expires or no other ATA or NFC request is asserted.

23.2.1.3 LPC RX/TX FIFO Registers

LPC uses a single FIFO that changes direction based on the RX/TX mode. Software controls direction change and flushes FIFO before changing directions. FIFO memory is 1024 bytes (256 x 32).

LPC FIFO is controlled by four 32-bit registers. These registers are located at an offset from IMMR of $\langle \text{lpc_addr_offset} \rangle + 0x0140$. Register addresses are relative to this offset. Therefore, the actual register address is $\text{IMMR} + \langle \text{lpc_addr_offset} \rangle + \text{register address}$.

Hyperlinks to the LPC FIFO registers are provided below:

[Section 23.2.1.3.1, “LPC RX/TX FIFO Data Word Register” \(0x0140\)](#)

[Section 23.2.1.3.2, “LPC RX/TX FIFO Status Register” \(0x0144\)](#)

[Section 23.2.1.3.3, “LPC RX/TX FIFO Control Register” \(0x0148\)](#)

[Section 23.2.1.3.4, “LPC RX/TX FIFO Alarm Register” \(0x014C\)](#)

23.2.1.3.1 LPC RX/TX FIFO Data Word Register

Offset 0x0140 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FIFO_Data_Word															
W																
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FIFO_Data_Word															
W																
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

X: Bit is not reset to a defined value.

Figure 23-18. LPC RX/TX FIFO Data Word Register

Table 23-19. LPC RX/TX FIFO Data Word Register Field Descriptions

Field	Description
FIFO_Data_Word	The FIFO data port. Reading from this location fetches data from the FIFO writing to this location writes data into the FIFO. During normal operation, the DMA controller moves data to and from this register. Note: Only full-word access is allowed. If all byte enables are not asserted when accessing this location, a FIFO error flag is generated.

23.2.1.3.2 LPC RX/TX FIFO Status Register

Offset 0x144 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W										ERR	UF	OF		FULL	ALARM	EMPTY
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 23-19. LPC RX/TX FIFO Status Register

Table 23-20. LPC RX/TX FIFO Status Register Field Descriptions

Field	Description
ERR	Error. The flag bit is essentially the logical OR of UF and OF bits and can be polled to detect any FIFO error. After clearing the offending condition, writing 1 to this bit clears the flag.
UF	Underflow. The flag indicates the read pointer has surpassed the write pointer. FIFO was read beyond empty. Resetting FIFO clears this condition; writing 1 to this bit clears the flag.
OF	Overflow. The flag indicates the write pointer has surpassed the read pointer. FIFO was written beyond full. Resetting FIFO clears this condition; writing 1 to this bit clears the flag.
FULL	FIFO full. A full indication tracks with FIFO state.
ALARM	This bit is set when the FIFO level is at or below (write)/above (read) the alarm watermark, as written according to the LPC RX/TX FIFO Alarm Register setting. This bit is cleared when the FIFO level is at or above (write)/below (read) the granularity watermark, as configured according to the LPC RX/TX FIFO Control Register setting. Setting this bit automatically signals the DMA engine to refill (write)/empty (read) the FIFO.
EMPTY	FIFO empty. An empty indication tracks with FIFO state.

23.2.1.3.3 LPC RX/TX FIFO Control Register

Offset 0x0148Access: User read/write

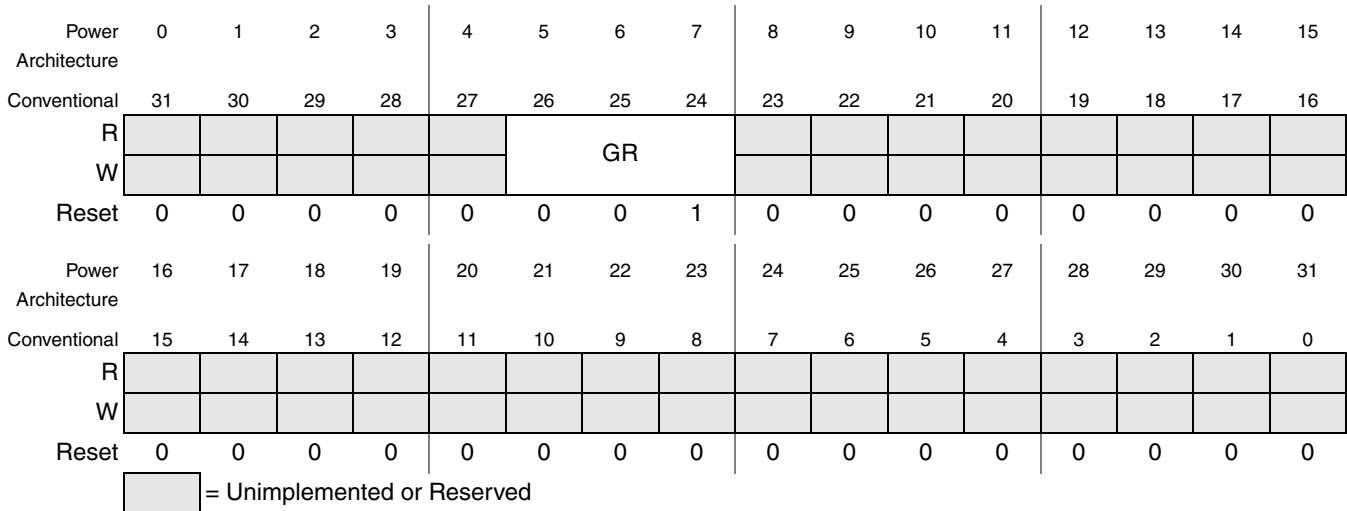


Figure 23-20. LPC RX/TX FIFO Control Register

Table 23-21. LPC RX/TX FIFO Control Register Field Descriptions

Field	Description
GR	Granularity. These bits control the high (write)/low (read) watermark point at which FIFO negates the alarm condition (i.e., a request for data). It represents the number of free bytes times four. 000 FIFO waits to become completely full (write)/empty (read) before stopping the data request. 001 FIFO stops the data request when only one word of space (write)/one word (read) remains.

23.2.1.3.4 LPC RX/TX FIFO Alarm Register

Offset 0x144 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 23-21. LPC RX/TX FIFO Alarm Register

Table 23-22. LPC RX/TX FIFO Alarm Register Field Descriptions

Field	Description
ALARM_W	<p>Alarm Watermark. Write these bits to set the low (write)/high (read) level watermark, which is the point where FIFO asserts a request for the DMA controller data to fill (write)/empty (read). The value is in bytes.</p> <p>For example, During a SCLPC write operation (R/\overline{W} bit of SCLPC Control Register) and an alarm watermark setting of 128, an alarm condition occurs when FIFO contains 128 bytes or less.</p> <p>After asserted, an alarm does not negate until a high (write)/ low (read) level mark is reached, as specified by the LPC RX/TX FIFO Control Register granularity bits.</p>

23.3 Functional Description

There are two primary modes of operation:

- Muxed
- Non-muxed

Within each mode, there is considerable flexibility to control the operation.

Each CS can be programmed to a different mode of operation (muxed, non-muxed, number of wait states, byte swapping, etc.).

In muxed mode, the same 32-bit local bus presents an address in an address tenure and data in a data tenure, in a muxed fashion (similar to PCI protocol).

Muxed mode provides an ALE during the address phase to capture the address. This mode requires external logic to latch the address during the address tenure. The level of the ALE can be programmed by the ALE bit.

An $\overline{\text{ACK}}$ input is provided and can be asserted to shorten (but not extend) wait states.

The LPC on MPC5121e provides an output enable signal, $\overline{\text{OE}}$, to achieve a complete glueless interface for most devices. $\overline{\text{OE}}$ is asserted one clock after the CS assertion if a read transaction occurs.

Muxed and non-muxed modes support a variety of device configurations and are configurable on a per CS basis. The read and write burst functionality is available in both modes. In non-muxed mode, an asynchronous burst (page mode burst) is also possible.

23.3.1 Non-Muxed Mode

In non-muxed mode, the 32-bit address/data bus is divided into address and/or data lines.

Table 23-23. Non-Muxed Mode Options

Data Size	Address Bus Width	Pins Used	Memory Size	Comments
8	32	40	4 GBytes	
16	26	42	128 MBytes	Short addressing
16	26	42	64 MBytes	Byte addressing
32	10	42	4 KBytes	Word addressing
32	10	42	1 KBytes	Byte addressing

NOTE

The 24-bit data width is not supported.

The total pin number also requires the addition of the control signals CS, $\text{R}/\overline{\text{W}}$, $\overline{\text{ACK}}$, $\overline{\text{OE}}$, $\overline{\text{TS}}$ (programmable), and TSIZ (programmable) where available.

Table 23-24. Internal Data and Address Bus Assignment to External Signals in Non-Muxed Mode

Signal Name	8-Bit Data Bus	16-Bit Data Bus		32-Bit Data Bus	
		Byte Addressing	Short Addressing	Byte Addressing	Word Addressing
AX[09:08]	0	address[25:24]	address[26:25]	address[9:8]	address[11:10]
AX[07:00]	address[31:24]	address[23:16]	address[24:17]	address[7:0]	address[9:2]
AD[31:24]	address[23:16]	address[15:8]	address[16:9]	data[31:24]	
AD[23:16]	address[15:8]	address[7:0]	address[8:1]	data[23:16]	
AD[15:8]	address[7:0]	data[15:8]		data[15:8]	
AD[7:0]	data[7:0]	data[7:0]		data[7:0]	

TSIZ bits can be enabled for every mode, but the TSIZ bits only make sense for the byte addressing mode. The low address bits, are not available in the short or word addressing mode. Only TSIZES of 1, 2, or 4 are supported.

TSIZ[1:0] are driven as follows:

01 = Transaction is 1 byte

10 = Transaction is 2 bytes

00 = Transaction is 4 bytes

Other values are invalid and should not be required by the external device.

Table 23-25 and Table 23-26 describes the various combinations of TSIZ. In addition also the influence of the swap bits - WS and RS of Chip Select X Configuration Register - is shown.

Table 23-25. Aligned Data Transfers for 32-Bit Data Bus Width

Swap WS or RS set	Transfer Size	TSIZ[1:0]	Address[1:0]	Data Lanes			
				AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
NO/YES ¹	1 byte	01	00	Data	—	—	—
			01	—	Data	—	—
			10	—	—	Data	—
			11	—	—	—	Data
NO	2 bytes	10	00	Data1	Data2	—	—
			10	—	—	Data1	Data2
YES			00	Data2	Data1	—	—
			10	—	—	Data2	Data1
NO	4 bytes	00	00	Data1	Data2	Data3	Data4
YES				Data4	Data3	Data2	Data1

Note: Data1 is most significant byte and Data2 (2 byte transfer) or Data 4 (4 byte transfer) is lowest significant byte of data word.

¹ Swap setting has no influence on byte transfers.

Table 23-26. Aligned Data Transfers for 16-Bit Data Bus Width

Swap WS or RS set	Transfer Size	TSIZ0	Address 0	Data Lanes	
				AD[15:8]	AD[7:0]
NO/YES ¹	1 byte	1	0	Data	—
			1	—	Data
NO	2 bytes	0	0	Data1	Data2
YES				Data2	Data1

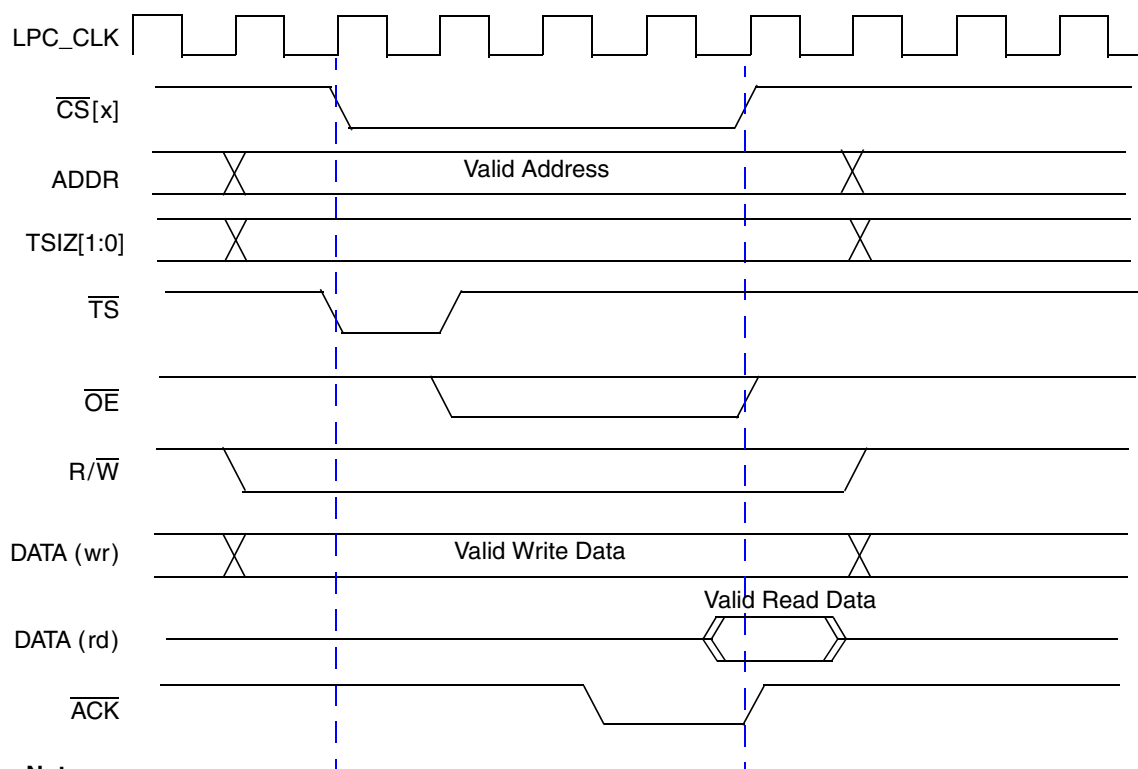
Note: Data1 is most significant byte and Data2 is lowest significant byte of data word.

¹ Swap setting has no influence on byte transfers.

Figure 23-22 shows a non-muxed transaction. Figure 23-23 and Figure 23-24 show non-muxed synchronous bursts transactions. Figure 23-25 and Figure 23-26 show non-muxed asynchronous burst (page mode) transactions. Detailed information about timing diagrams and the influence of register settings can be found in the datasheet.

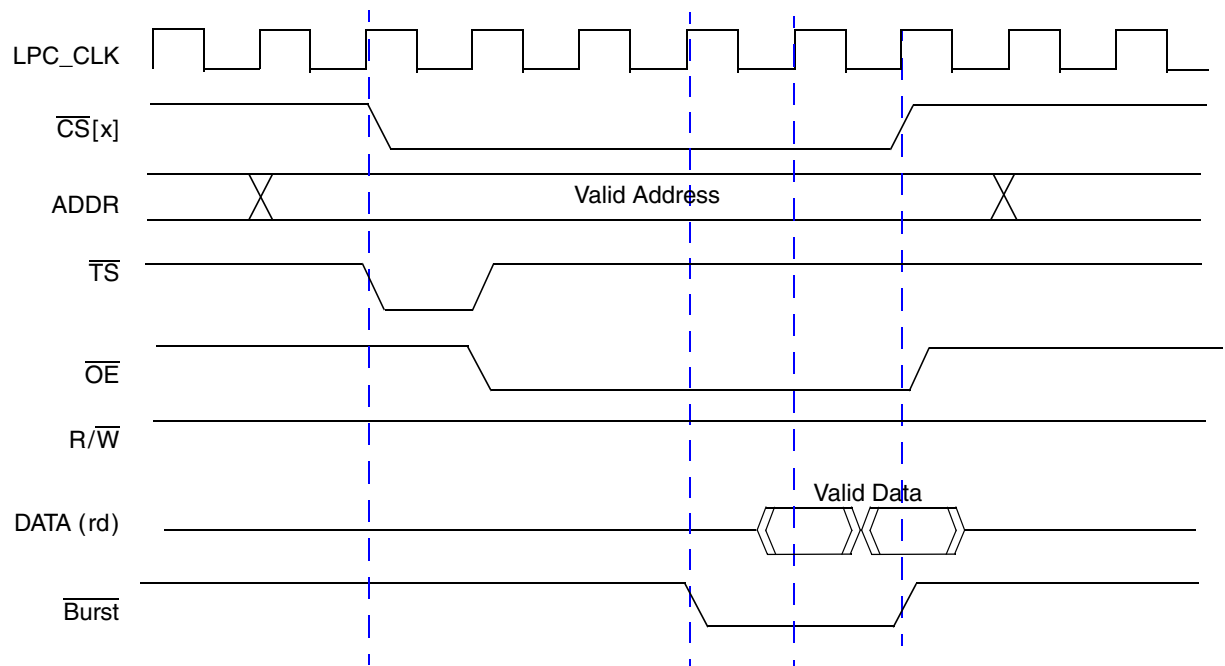
NOTE

In the following five diagrams, deadcycle and holdcycle are each set to 0.

**Note:**

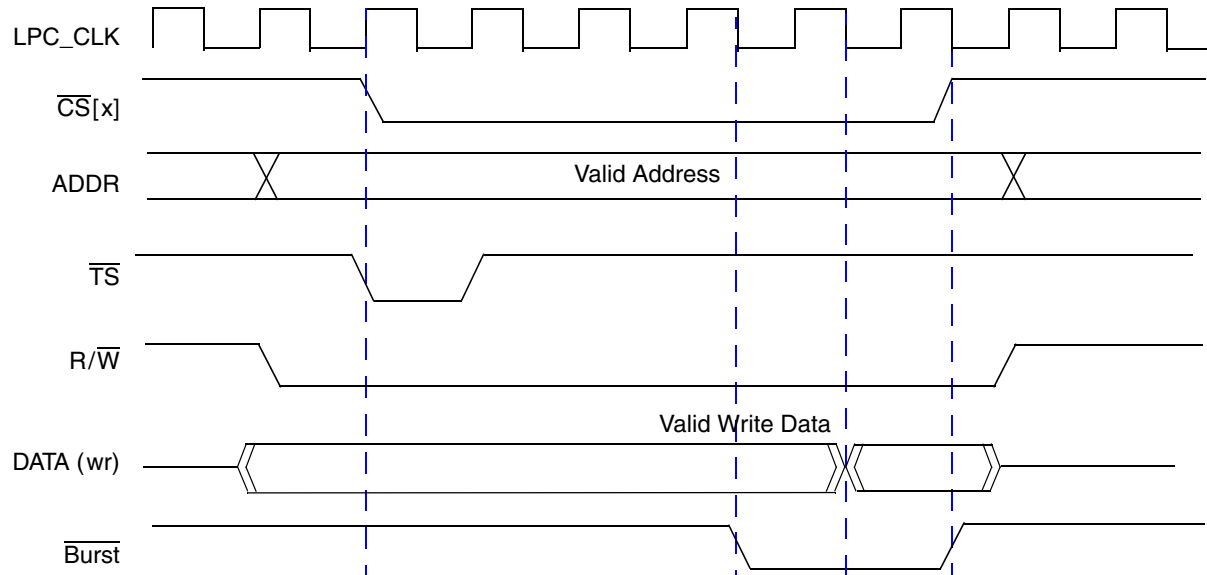
\overline{ACK} can shorten the CS pulse width.
 Related to the holdcycle setting, write data can stay on the bus longer.
 This diagram represents a wait state setting of 2.

Figure 23-22. Timing Diagram—Non-Muxed Mode



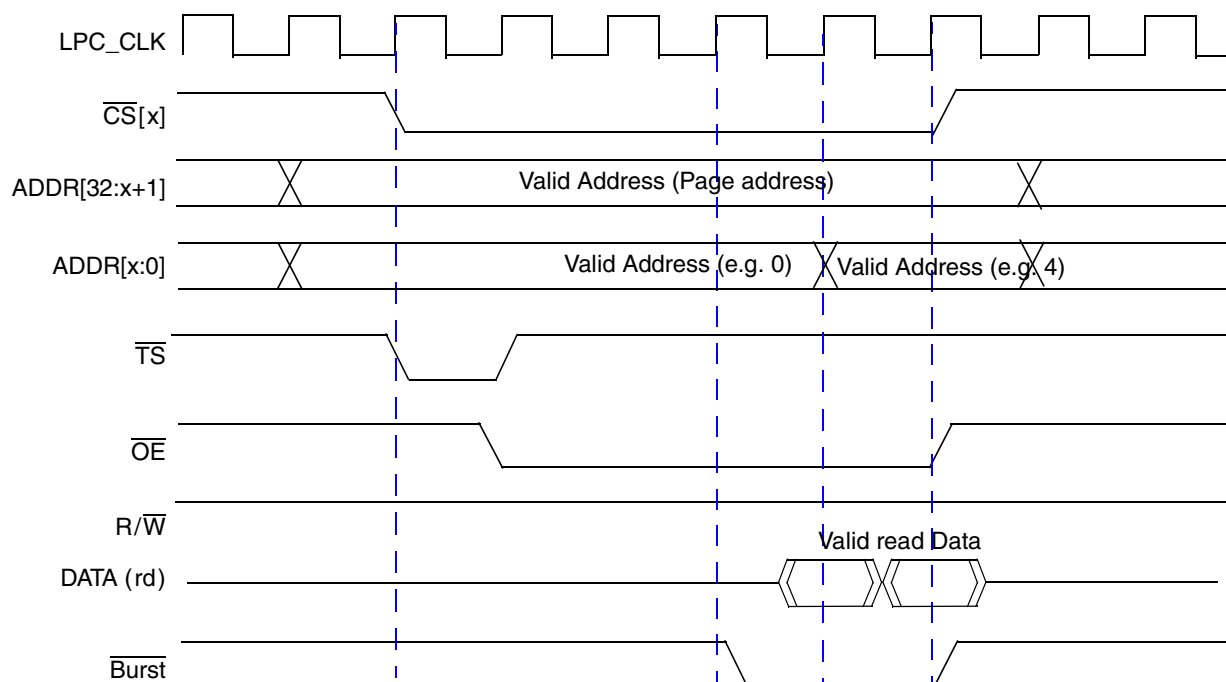
Note: This diagram represents a wait state setting of 1.

Figure 23-23. Timing Diagram—Non-Muxed Synchronous Read Burst



NOTES:
This diagram represents a wait state setting of 1.

Figure 23-24. Timing Diagram—Non-Muxed Synchronous Write Burst



Notes:

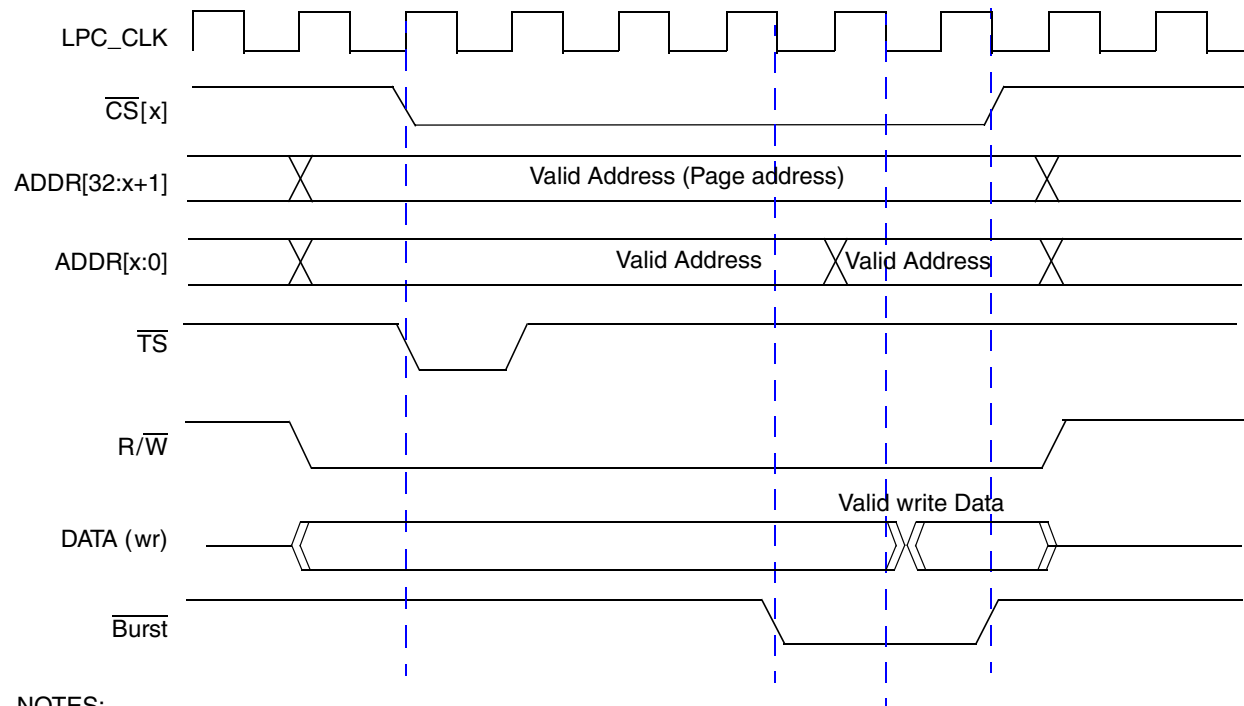
If the transaction is initiated by a master on the CSB, the address wraps around, if necessary, at cache level boundaries (e.g., 0x20; 0x40...)

If the transaction is initiated by the SCLPC module, the address continuously increases.

This diagram represents a wait state setting of 1.

The address changes at the rising edge of the LPC clock.

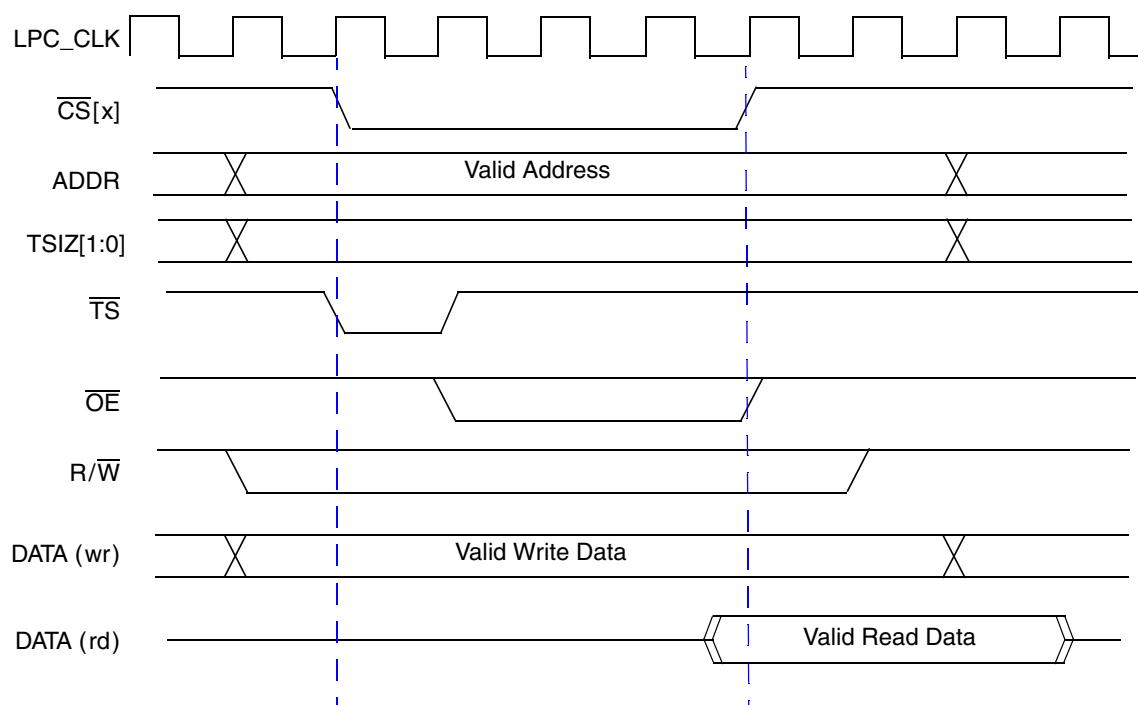
Figure 23-25. Timing Diagram—Non-Muxed Asynchronous Read Burst



NOTES:

- If the transaction is initiated by a master on the CSB, the address wraps around, if necessary, at cache level boundaries.
- If the transaction is initiated by the SCLPC module, the address continuously increases with no address wrap-around.
- This diagram represents a wait state setting of 1.
- The address changes at the rising edge of the LPC clock.

Figure 23-26. Timing Diagram—Non-Muxed Asynchronous Write Burst

**Note:**

- This diagram represents a holdcycle setting of 1.
- This diagram represents a deadcycle setting of 2.
- This diagram represents a wait state setting of 2.

Figure 23-27. Timing Diagram—Non-Muxed Mode with Different Timing Settings

In Non-Muxed mode the address and data are driven simultaneously on the external AD/AX bus. A single dedicated R/ \overline{W} pin is driven to indicate read or write. An individually dedicated CS pin is driven low while an external access is active.

Wait states are programmable and simply select how many LPC clocks the CS pin remain asserted. Separate values are available for read cycles versus write cycles. These values can be combined to create extremely long (up to 16 bits) cycles. Byte lane swapping is separately programmable between reads versus writes and can be used to perform endian conversions. The 24-bit data width is not supported.

Devices can be marked as read-only or write-only by setting a control bit in the appropriate LPC register. Attempted accesses in violation of this setting are prevented. Each CS pin can be individually enabled/disabled and the entire LPC module has a master enable bit. No software reset bit is provided or needed.

The non-muxed mode requires no external logic for interfacing to simple devices such as flash ROM, EEPROM or SRAM. It is faster than the muxed mode because data and address are provided in a single tenure.

23.3.2 Muxed Mode

In muxed mode, the addresses and data are muxed using dual tenure. First, the address is put on the shared address/data bus and ALE is asserted. The data is then driven when the chip select is asserted. Different modes of data sizes can be configured, as shown in [Table 23-27](#).

Table 23-27. Muxed Mode Options

Data Size	Address Size	Comments
8	32	—
16	31	Short addressing; AD 0 is connected to internal address 1. Internal address 0 is not brought out.
16	32	Byte addressing
32	30	Word addressing; AD 0 is connected to internal address 2. Internal address 0 and 1 is not brought out.
32	32	Byte addressing

NOTE

The 24-bit data width is not supported.

Table 23-28. Internal Data and Address Bus Assignment to External Signals in Muxed Mode

Signal Name	8-Bit Data Bus			16-Bit Data Bus			32-Bit Data Bus		
	Addr Phase	Isolation	Data Phase	Addr Phase	Isolation	Data Phase	Addr Phase	Isolation	Data Phase
AX[03]	1	1	$\overline{\text{TS}}$	1	1	$\overline{\text{TS}}$	1	1	$\overline{\text{TS}}$
AX[02:01]	TSIZ[1:0]			TSIZ[1:0]			TSIZ[1:0]		
AX[00]	ALE ¹	!ALE	!ALE	ALE	!ALE	!ALE	ALE	!ALE	!ALE
AD[31:24]	address [31:24]	address [31:24]	0	address [31:24]	address [31:24]	0	address [31:24]	address [31:24]	data [31:24]
AD[23:16]	address [23:16]	address [23:16]	0	address [23:16]	address [23:16]	0	address [23:16]	address [23:16]	data [23:16]
AD[15:8]	address [15:8]	address [15:8]	0	address [15:8]	address [15:8]	data [15:8]	address [15:8]	address [15:8]	data [15:8]
AD[7:0]	address [7:0]	address [7:0]	data [7:0]	address [7:0]	address [7:0]	data [7:0]	address [7:0]	address [7:0]	data [7:0]

¹ ALE represents the programmable value of the ale bit (csboot/csx configuration registers).

Table 23-29. Internal Data and Address Bus Assignment to External Signals in Muxed Mode (Short/Word Addressing)

Signal Name	8-Bit Data Bus			16-Bit Data Bus			32-Bit Data Bus		
	Addr Phase	Isolation	Data Phase	Addr Phase	Isolation	Data Phase	Addr Phase	Isolation	Data Phase
AX[03]	1	1	TS	1	1	TS	1	1	TS
AX[02:01]	TSIZ[1:0]			TSIZ[1:0]			TSIZ[1:0]		
AX[00]	ALE ¹	!ALE	!ALE	ALE	!ALE	!ALE	ALE	!ALE	!ALE
AD[31]	address 31	address 31	0	0	0	0	0	0	D31
AD[30]	address 30	address 30	0	address 31	address 31	0	0	0	D30
AD[29:24]	address [29:24]	address [29:24]	0	address [30:25]	address [30:25]	0	address [31:26]	address [31:26]	data [29:24]
AD[23:16]	address [23:16]	address [23:16]	0	address [24:17]	address [24:17]	0	address [25:18]	address [25:18]	data [23:16]
AD[15:8]	address [15:8]	address [15:8]	0	address [16:9]	address [16:9]	data [15:8]	address [17:10]	address [17:10]	data [15:8]
AD[7:0]	address [7:0]	address [7:0]	data [7:0]	address [8:1]	address [8:1]	data [7:0]	address [9:2]	address [9:2]	data [7:0]

¹ ALE represents the programmable value of the ale bit (csboot/csx configuration registers).

An ALE signal is asserted during this address tenure. ALE level is programmable to be low or high. The dedicated R/W output is also driven with ALE (and throughout the cycle). One clock after the ALE negates, the appropriate CS pin asserts (low) and the AD bus enters the data tenure (isolation cycle). The CS pin and this data tenure remain active until the programmed wait states expire or the device responds with an ACK assertion. ACK polarity is active low, but can be programmed to be ignored. The data tenure can contain up to the full 32-bit width. However, the data width is programmable to support dynamic bus-sized transactions.

The muxed mode requires external logic to latch the address during the address tenure and to decode bank selects if they are encoded. This mode is slower than the non-muxed mode because data and address are muxed in time. The supported address space is limited by the 30 address lines. In muxed mode, LPC can access up to 4 GBytes of data.

23.3.2.1 Address Tenure

The address is presented on the corresponding AD bus bits, which total 32 bits (AD[31:0]).

The TSIZ bits appear on AX[34] (TSIZ most significant bit) to AX[33] (TSIZ least significant bit). These bits are calculated and driven by the LPC based on the internal byte lane enables on the IP bus.

NOTE

Only TSIZes of one, two, or four are supported.

TSIZ [1:0] are driven as follows:

01 = Transaction is 1 byte

10 = Transaction is 2 bytes

00 = Transaction is 4 bytes

NOTE

Other values are invalid and should not be required by the external device.

Table 23-30 and Table 23-31 describe the various combinations of TSIZ, address, and byte lanes for a 32-bit and 16-bit wide data bus. In addition also the influence of the swap bits - WS and RS of Chip Select X Configuration Register - is shown.

Table 23-30. Aligned Data Transfers for 32-Bit Data Bus Width

Swap WS or RS set	Transfer Size	TSIZ[1:0]	Address[1:0]	Data Lanes			
				AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
NO/YES ¹	1 byte	01	00	Data	—	—	—
			01	—	Data	—	—
			10	—	—	Data	—
			11	—	—	—	Data
NO	2 bytes	10	00	Data1 ²	Data2	—	—
			10	—	—	Data1	Data2
YES			00	Data2	Data1	—	—
			10	—	—	Data2	Data1
NO	4 bytes	00	00	Data1	Data2	Data3	Data4
YES				Data4	Data3	Data2	Data1

¹ Swap setting has no influence on byte transfers.

² Data1 is most significant byte and Data2 (2 byte transfer) or Data 4 (4 byte transfer) is lowest significant byte of data word.

Table 23-31. Aligned Data Transfers for 16-Bit Data Bus Width

Swap WS or RS set	Transfer Size	TSIZ0	Address 0	Data Lanes	
				AD[15:8]	AD[7:0]
NO/YES ¹	1 byte	1	0	Data	—
			1	—	Data
NO	2 bytes	0	0	Data1 ²	Data2
YES				Data2	Data1

¹ Swap setting has no influence on byte transfers.

² Data1 is most significant byte and Data2 is lowest significant byte of data word.

The ALE signal is active low or high, depending on the ALE bit setting, and remains asserted for several external LPC bus clocks, depending on the ALEN bit field setting. Any external latch should be transparent when active.

23.3.2.2 Data Tenure

During data tenure, the following occurs:

- When a write to the device occurs, the LPC drives the indicated AD bits.
- When a read occurs, the indicated AD bits are tri-stated by the LPC.

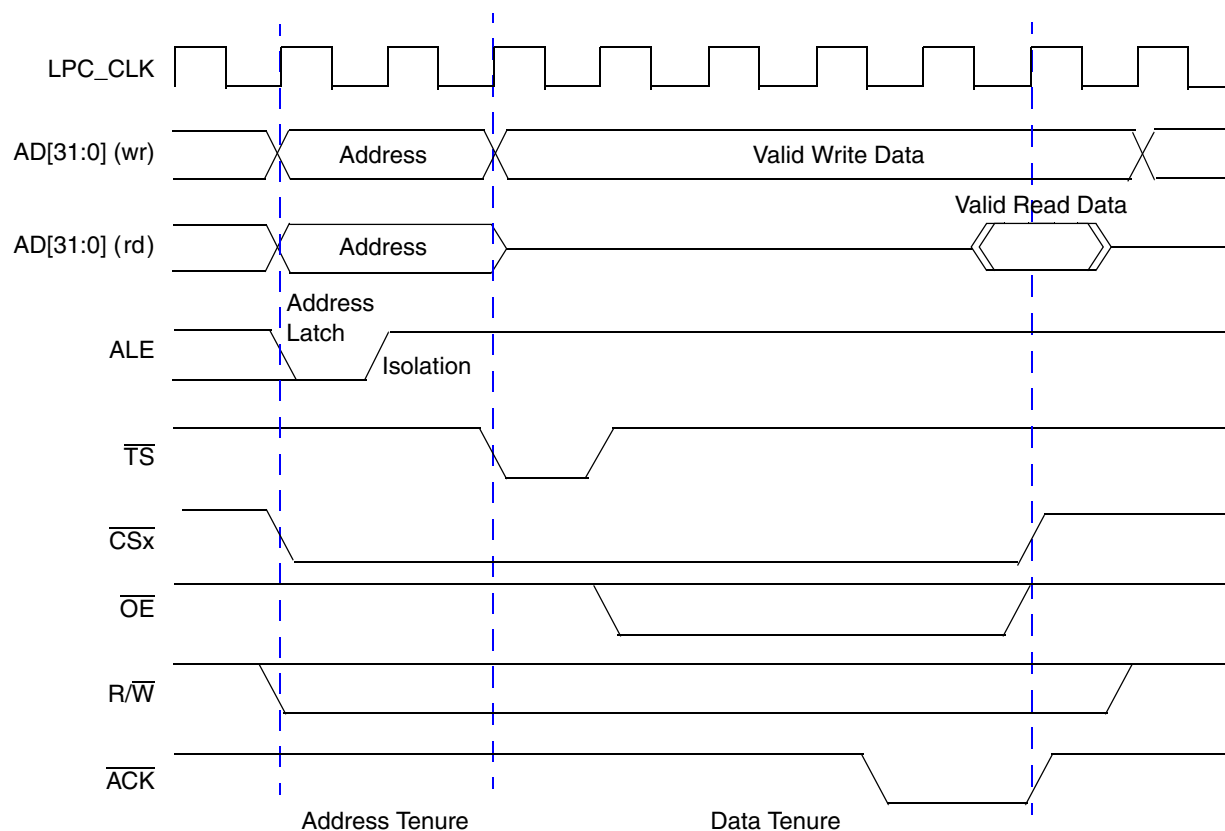
NOTE

AD[0] is treated as the least significant data bit. Any unused data bits (as indicated by the data size field in the associated control register) are driven low by the LPC. Therefore, they should not be driven by the device or glue chip.

The ACK input signal is internally synchronized to the internal clock. At the first LPC clock edge where the ACK input is detected as asserted, the LPC terminates the transaction and releases the bus on the next LPC bus clock. [Figure 23-28](#) shows a muxed transaction-type timing diagram. [Figure 23-28](#), [Figure 23-29](#), and [Figure 23-30](#) show muxed burst transactions. Detailed information about timing diagrams and the influence of register settings can be found in the datasheet.

NOTE

In the following diagrams, deadcycle and holdcycle are each set to 0.

**NOTES:**

$\overline{\text{ACK}}$ can shorten the CS pulse width.

ALE can be active high or low, depending on the ale bit of the csboot/csx configuration register.

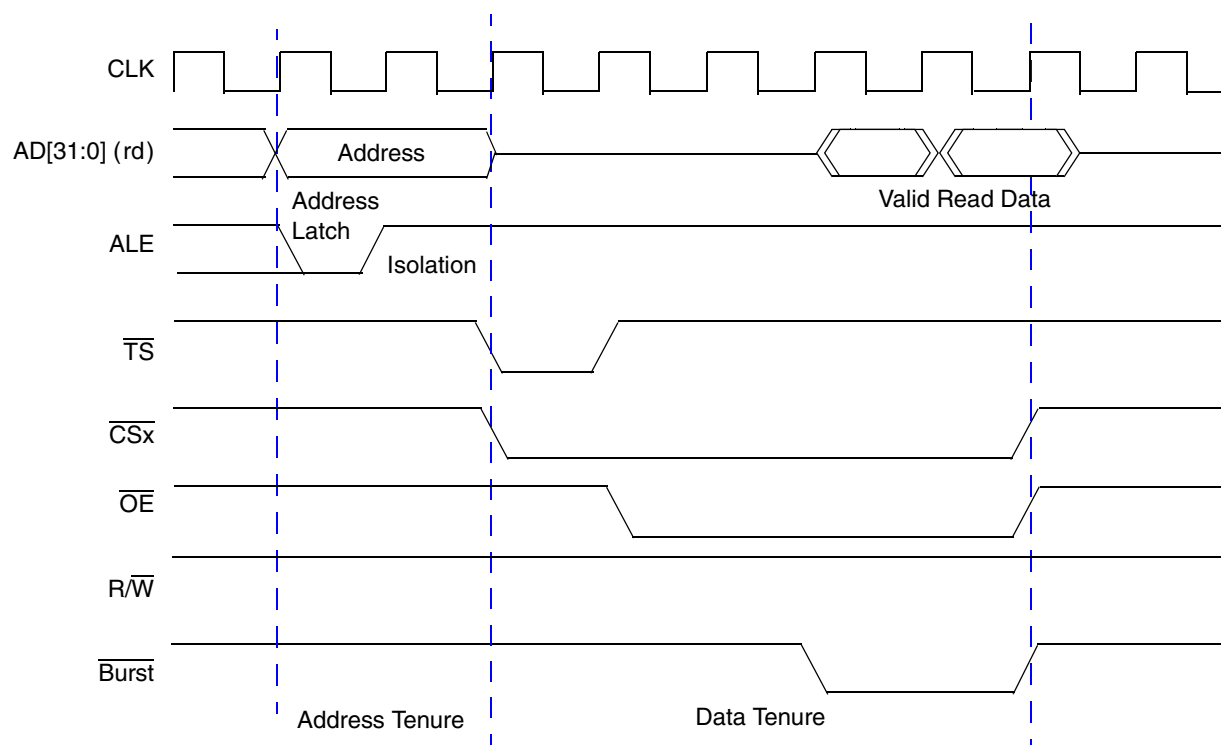
This diagram shows an active low scenario, which means the ALE bit is set to 0.

The length of the address latch and of the isolation cycle can be configured by the ALLEN bit of the CSBOOT/CSX configuration register. This diagram shows a setting of 0.

This diagram represents a wait state setting of 3.

Address Latch bit are set to zero. CS is asserted together with ALE.

Figure 23-28. Timing Diagram – Muxed Mode



NOTES:

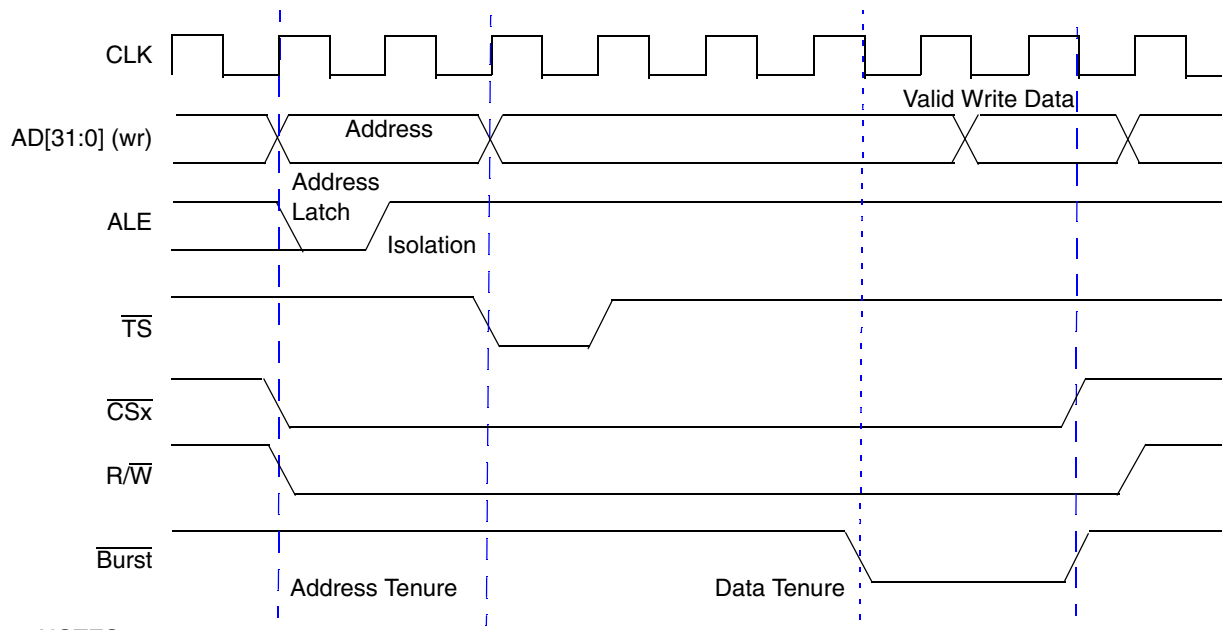
ALE can be active high or low. This diagram shows an active low scenario.

The length of the address latch and of the isolation cycle can be configured by the ALEN bit of the CSBOOT/CSX configuration register. This diagram shows a setting of 0.

This diagram represents a wait state setting of 1.

Address Latch bit are set to one. CS is asserted after the isolation cycle.

Figure 23-29. Timing Diagram – Muxed Synchronous Read Burst



NOTES:

ALE can be active high or low. This diagram shows an active low scenario.

The length of the address latch and of the isolation cycle can be configured by the ALEN bit of the CSBOOT/CSX configuration register. This diagram shows a setting of 0.

This diagram represents a wait state setting of 1.

Address Latch bit are set to zero. CS is asserted together with ALE.

Figure 23-30. Timing Diagram—Muxed Synchronous Write Burst

23.3.2.3 Boot Configuration

After reset, the core can fetch the first instruction from the LPC. The chip select boot (CSBoot) is dedicated for this purpose. CSBoot and CS0 are physically the same pins. The difference is that CSBoot is affected by the reset configuration and is enabled after reset.

Several options are available for this purpose:

- Muxed or non-muxed mode
- Byte or short word addressing
- Data size can be 8, 16, or 32 bits

23.3.2.4 Chip Selects Configuration

All chip selects (CS0-7) have the same functionality. Only one CS can be active at any given time. If an address hit is located in multiple CS windows, only the CS with the highest priority becomes active. The CS with the lowest number has the highest priority (CS0 = highest priority, CS7 = lowest priority).

CSBoot and CS0 are identical with the exception of their control registers, contained in the XLBMEN register map (LocalPlus Boot/CS0-7 Access Window Registers (LPBAW/LPCSxAW) and LocalPlus CS0-7 Access Window Registers (LPCSxAW)); see [Section 23.3.2.3, “Boot Configuration”](#). CSBoot and CS0 are physically the same pins. The difference is that CSBoot is affected by the reset configuration and is the only enabled chip select after reset.

To change from CSBoot to CS0, the CS0 start and stop addresses must be configured and CSBoot must be disabled at the same time CS0 is enabled.

Deadcycles from 0 to 3 can be added to any CS read access and occur in addition to any cycles that already exist. The chip select deadcycle control register configures deadcycles.

Holdcycles from 0 to 3 can be added to any CS write access and occur in addition to any cycles that already exist.

Burst mode operations are supported on all CSs and all modes and can be configured by the chip select burst control register.

The following features are valid for all CS's and for both modes (non-muxed and muxed):

- Supports 8-, 16-, and 32-bit data bus width
- Supports dynamic bus sizing, which means read and write transactions greater than the defined port size are possible
- Transactions less than the defined port size are supported only if the device can decode the Tsiz[1:0] bits, which indicate the current transaction size.
- Supports code execution
 - The e300 processor can execute code from the LP bus from all CS
- Supports burst access (read and write)
 - Synchronous burst

Asynchronous burst mode (Page mode) is supported in non-muxed mode.

23.3.3 SCLPC Interface

The SCLPC interface, together with the DMA engine, can be used to transfer data between an external device and the DRAM without the usage of the e300 processor.

SCLPC controls the data transfer between the LPC RX/TX FIFO and the external device. The external transfer behavior is defined by the CSBoot/CS[x] configuration, chip select burst control, chip select deadcycle, and chip select holdcycle registers. The supported transfer sizes are limited, by BPT bit field setting, to 1, 2, 4, 8, 16, 24, 32, 40, 48, or 56 bytes only. Burst transfers could be generated with transfer size settings greater as 4.

The SCLPC controller supports half-duplex operation (transmit or receive) only. If software configures a transmit packet, the packet must be complete before a receive operation can be configured and started. The length of one transfer packet is defined with the Packet Size bit field of the SCLPC Packet Size Register. A transfer packet is split down into smaller bytes-per-transfer (BPT) slice. A BPT slice cannot be interrupted by a direct e300 access to an external device. A BPT slice needs to be finished first before the e300 access goes onto the external bus.

23.3.3.1 SCLPC Programming

The device specific behavior (non-/muxed, burst, wait cycles - [Table 23-3](#) through [Table 23-10](#)) needs to be programmed before the SCLPC is setup.

Following steps are to setup the SCLPC controller together with LPC RX/TX FIFO:

- Configure DMA to transfer data between LPC RX/TX FIFO and DRAM
 - The maximum Inner Minor Byte Count setting, working together with the LPC RX/TX FIFO, is 32 bytes
- Configure granularity watermark to 28. This is done by setting the GR bit field ([Table 23-21](#)) to 7
- Configure alarm watermark (ALARM_W bit field ([Table 23-22](#))). The setting is depending on the system load.
- Configure start address, chip select, direction (read/write), address increment and bytes-per-transfer ([Table 23-12](#) and [Table 23-13](#)). In case of a read transfer from an external device set Flush bit. It doesn't need to be set for a write transfer.
- Setup packet size. Don't set Restart bit during this.
- Enable SCLPC interrupt by setting ME and NIE bits of SCLPC Enable Register
- Reset SCLPC controller and FIFO.
- Enable SCLPC controller and FIFO by clearing SCLPC controller and FIFO reset bits.

Following steps are to start SCLPC transfers:

- Set Restart bit of SCLPC Packet Size Register
 - In case of read transaction the SCLPC controller starts to fill the LPC RX/TX FIFO
 - In case of write transaction SCLPC controller is waiting until BPT size data is written by the DMA into the LPC RX/TX FIFO
- Start LPC DMA request within the DMA module

The transfer is finished when

- In case of a read transfer, the DMA signals via interrupt that the LPC TCD is finished
- In case of a write transfer, the SCLPC signals via interrupt that the transfer is finished

23.3.4 Programmer's Model

[Table 23-3](#) through [Table 23-10](#) describe the registers and bit meanings for configuring CS operation in detail. There are eight identical CS configuration registers, one for each CS output. However, the CSBoot ROM configuration register has active defaults for use by BOOTROM on CS0. All other configuration registers are disabled at powerup and require software intervention before the corresponding CS operates. The chip select control register is the enable register and the chip select status register serves as a status register. The chip select burst control register enables burst mode and deadcycles are configured by the chip select deadcycle control register.

NOTE

The address range registers for each CS reside in the XLBMEN register set rather than in the LPC register set. See [Section 23.3.2.3, "Boot Configuration"](#).

Chapter 24

MBX Graphics Controller

24.1 Introduction

This module describes the PowerVR Series 3 MBX Lite 3D graphics module designed for mobile applications.

The MBX module is the intellectual property of Imagination Technologies and is licensed for use in the MPC5121e. This document does not describe the internal registers or operation of the MBX module due to licensing restrictions.

Therefore, developers who wish to use the MBX module need to use an existing graphics driver for this block.

Freescale makes an MBX driver available for their Linux OS that implements the OpenGL ES standard.

Additional drivers are expected to be available from other third party vendors.

24.1.1 Overview

The MBX module refers to the entire MBX block including the MBX-lite, VGP-lite cores, and all the interconnecting gaskets. The MBXLITE core refers to the MBX-lite and VGP_lite, both licensed from Imagination Technologies, and the various interconnecting gaskets used to interface the MBXLITE to the MPC5121e.

The MBX module can be accessed by E300 or DMA2 and there is a memory bus connecting to DDR.

MBXLITE core always works in little endian mode. There is a byte swapping logic inside MBX module, which the customers can turn on/off to make sure that the data communicated between MBXLITE core and E300/DMA2 can be understood correctly by each other.

24.1.1.1 MBXLITE Core

The PowerVR MBXLITE Graphics Core is a deferred rendering device using tile based display lists. MBX Lite operates upon 3D scene data (sent as batches of triangles) which are transformed and lit either by the CPU or by the Vertex Geometry Processor (VGP Lite).

24.1.2 Features

The MBX Lite core has the following 3D features:

- Deferred texturing

- Screen tiling
- Flat and ground shading
- Perspective correct texturing
- Specular high lights
- Floating point Z buffer
- 32-bit ARGB internal rendering and layer buffering
- Full-tile blend buffer
- Z load/store mode
- Per vertex fog
- 16 bit RGB textures
- 32 bit RGB textures
- YUV 422 textures
- PVR-TC compressed textures
- 1-bit textures for text acceleration
- Point, bilinear, trilinear, and anisotropic filtering
- Full range of OGL/D3D blend modes
- Dot3 bump mapping
- Alpha test
- Full-scene anti-aliasing
- 2Dvia3D
- Configurable YUV 420 textures

The 2D features are:

- ROP2, 3, 4 Support (including AA Text)
- Source, mask, and pattern from system or frame buffer memory
- Alpha blending (per-pixel and global)
- Colour key
- Input Formats:
 - 1, 2, 4, 8 palletised
 - 4-Bit alpha
- Input and output formats: (A)RGB
 - 3:3:2
 - 4:4:4:4
 - 5:5:5
 - 1:5:5:5
 - 5:6:5
 - 8:8:8
 - 8:8:8:8

- Stride up to 2048 pixels
- Up to four clipping rectangles
- Stretch BLTs
- 90-Degree BLT rotation

24.2 DMA operation

If you want to use the DMA engine of the chip to transfer data to MBX slave port, you first need to set up an appropriate DMA TCD and then set the DMA Set Enable Request register to the physical channel number of mbx (31). Then the DMA engine starts to work.

The MBXLITE Core continues asserting the DMA channel request signal to the DMA engine when there are more than 16 FIFO entries (each 32-bit wide) left in MBX slave port internal FIFO. The round robin scheme arbitrates between CPU and DMA accesses to the MBXLITE Core.

For details on how to configure the DMA engine, see [Section Chapter 11, “Direct Memory Access \(DMA\)”](#).

24.3 Clocking Architecture of the MBXLITE Core

The MBXLITE core has been designed to use two core clocks. Both MBXLITE core clocks must run at the same frequency. The MPC5121e drives these core clocks with the MBX_3D_CLK signal.

The system bus clock, MBX_BUS_CLK, is a derivative of the coherent system bus clock. The MBX_BUS_CLK signal must be programmed to a multiple of 1x, 2x, 3x, or 4x times the MBX_3D_CLK signal.

Chapter 25

MSCAN

25.1 Introduction

This module contains four identical and independent MSCAN Controllers.

The module is a communication controller implementing the CAN 2.0 A/B protocol as defined in the BOSCH specification dated September 1991. To fully understand the MSCAN specification, read the Bosch specification first to familiarize yourself with terms and concepts contained within this document.

The CAN protocol was primarily designed as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth.

MSCAN uses an advanced buffer arrangement resulting in a predictable real-time behavior and simplifies the application software.

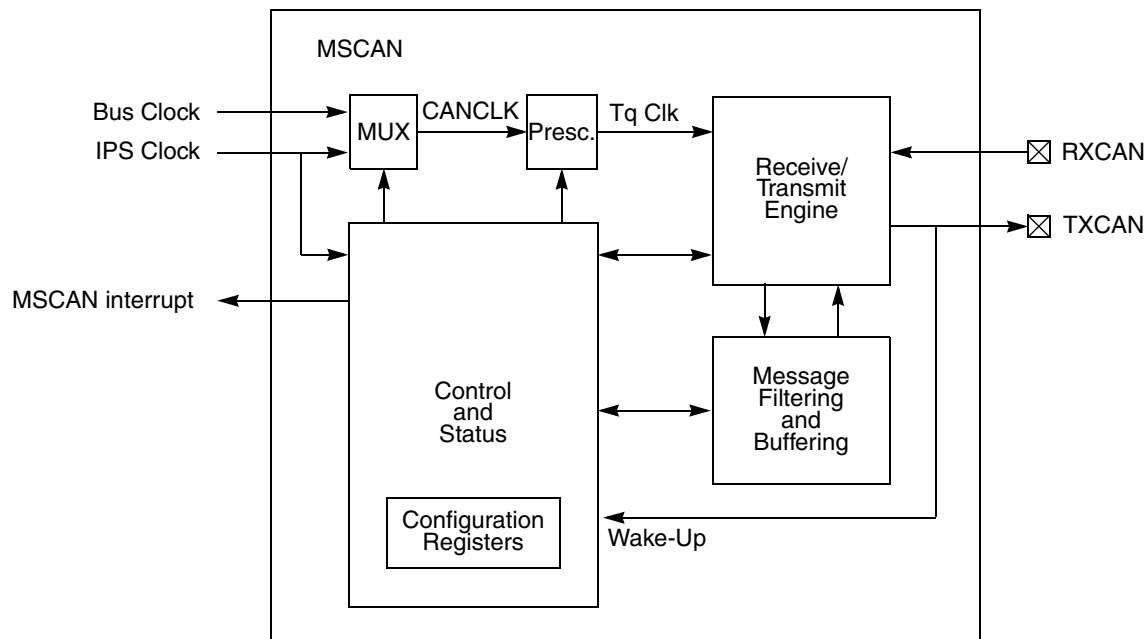


Figure 25-1. MSCAN Block Diagram

25.1.1 Features

The basic features of the MSCAN are:

- Implementation of the CAN protocol – Version 2.0A/B
 - Standard and extended data frames
 - 0 – 8 bytes data length
 - Programmable bit rate up to 1 Mbps (Depending on actual bit timing and clock jitter of PLL)
 - Support for remote frames
 - Five receive buffers with FIFO storage scheme
- Three transmit buffers with internal prioritization using a local priority concept
- Flexible maskable identifier filter supports two full-size extended identifier filters (two 32-bit) or four 16-bit filters or eight 8-bit filters
- Programmable wake-up functionality
- Programmable loop back mode supports self-test operation
- Programmable listen-only mode for monitoring of CAN bus
- Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states (warning, error passive, bus-off)
- Programmable clock source. See [Chapter 5, “Clocks and Low-Power Modes”](#).
- Internal timer for time-stamping of received and transmitted messages
- Three low power modes: sleep, power down and MSCAN enable
- Programmable bus-off recovery functionality
- Global initialization of configuration registers

25.2 External Signal Description

The MSCAN uses two external pins. In the module, the MSCAN pins are shared with other functionality and can be available at four different groups of pins. The configuration of the pin-muxing is controlled by the Port Configuration Register.

25.2.1 CAN Receiver Input Pins

Table 25-1. Signal Properties

Name	Port	Function	I/O
CAN1_RX	Input	MSCAN1 receiver input pin	I
CAN1_TX	Output	MSCAN1 transmitter output pin. The can1_tx output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O
CAN2_RX	Input	MSCAN2 receiver input pin	I
CAN2_TX	Output	MSCAN2 transmitter output pin. The can2_tx output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O
IRQ1_B	Input	MSCAN3 receiver input pin	I
IRQ0_B	Output	MSCAN3 transmitter output pin. The irq0_b output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O
PATA_ISOLATE	Output	MSCAN3 transmitter output pin. The pata_isolate output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O
J1850_RX	Input	MSCAN4 receiver input pin	I
J1850_TX	Output	MSCAN4 transmitter output pin. The j1850_tx output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O
I2C2_SDA	Input	MSCAN4 receiver input pin	I
I2C2_SCL	Output	MSCAN4 transmitter output pin. The i2c2_scl output pin represents the logic level on the CAN bus: 0 = Dominant state 1 = Recessive state	O

CAN1_RX, CAN2_RX, IRQ1_B, J1850_RX, and I2C2_SDA are the MSCAN receiver input pins. MSCAN4 can be configured to select J1850_RX or I2C2_SDA as input pin.

25.2.2 CAN Transmitter Output Pins

CAN1_TX, CAN2_TX, IRQ0_B, PATA_ISOLATE, J1850_TX, and I2C2_SCL are MSCAN transmitter output pins. MSCAN3 can be configured to select IRQ0_B or PATA_ISOLATE as output pin. MSCAN4

can be configured to select J1850_TX or I2C2_SCL as output pin. These pins represents the logic level on the CAN bus:

0 = Dominant state

1 = Recessive state

25.2.3 CAN System

A typical CAN system with MSCAN is shown in [Figure 25-2](#). Each CAN station is connected physically to the CAN bus lines through a transceiver device. The transceiver is capable of driving the large current needed for the CAN bus and has current protection against defective CAN or defective stations.

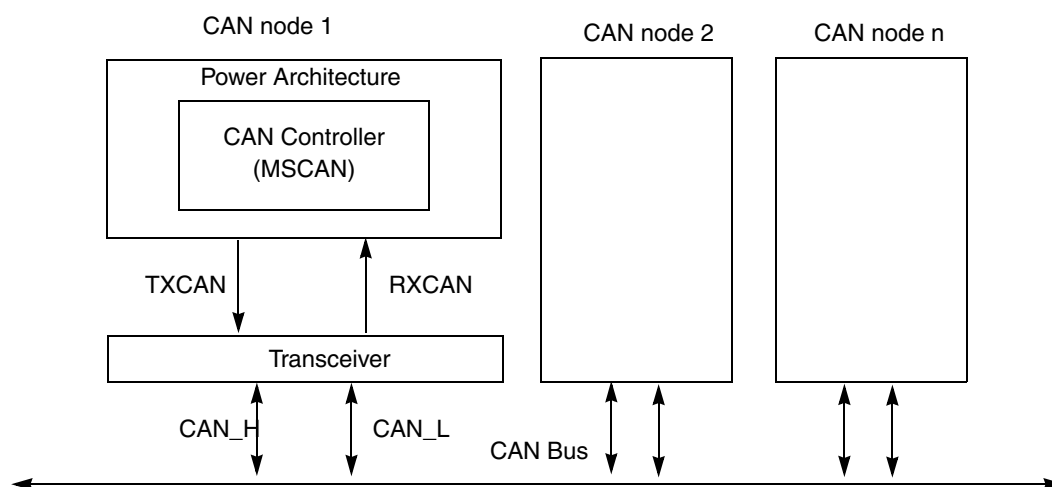


Figure 25-2. MSCAN System

25.3 Memory Map and Register Definition

The MPC5121e contains four independent MSCAN Controllers with identical register sets. The register sets have different base addresses. However, the registers in each set have the same offsets with respect to their base addresses.

Table 25-2. Block Memory Map

Offset	Register ¹	Access	Section/Page
General Registers			
0x00	MSCAN Control Register 0(CANCTL0)	R/W	25.3.2.1/25-7
0x01	MSCAN Control Register 1 (CANCTL1)	R/W	25.3.2.2/25-10
0x04	MSCAN Bus Timing Register 0 (CANBTR0)	R/W	25.3.2.3/25-12
0x05	MSCAN Bus Timing Register 1 (CANBTR1)	R/W	25.3.2.4/25-13
0x08	MSCAN Receiver Flag Register (CANRFLG)	R/W	25.3.2.5/25-14
0x09	MSCAN Receiver Interrupt Enable Register (CANRIER)	R/W	25.3.2.6/25-17
0x0C	MSCAN Transmitter Flag Register (CANTFLG)	R/W	25.3.2.7/25-19
0x0D	MSCAN Transmitter Interrupt Enable Register (CANTIER)	R/W	25.3.2.8/25-20
0x10	MSCAN Transmitter Message Abort Control (CANTARQ)	R/W	25.3.2.9/25-21

Table 25-2. Block Memory Map (continued)

Offset	Register ¹	Access	Section/Page
0x11	MSCAN Transmitter Message Abort Control (CANTAACK)	R	25.3.2.10/25-22
0x14	MSCAN Transmit Buffer Selection (CANTBSEL)	R/W	25.3.2.11/25-23
0x15	MSCAN Identifier Acceptance Control Register (CANIDAC)	R/W	25.3.2.12/25-24
0x19	MSCAN MISC Register (CANMISC)	R/W	25.3.2.13/25-25
0x1C	MSCAN Receive Error Counter Register (CANRXERR)	R	25.3.2.14/25-26
0x1D	MSCAN Transmitter Error Counter Register (CANTXERR)	R	25.3.2.15/25-27
0x20	MSCAN Identifier Acceptance Registers (CANIDAR0)	R/W	25.3.2.16/25-28
0x21	MSCAN Identifier Acceptance Registers (CANIDAR1)	R/W	25.3.2.16/25-28
0x24	MSCAN Identifier Acceptance Registers (CANIDAR2)	R/W	25.3.2.16/25-28
0x25	MSCAN Identifier Acceptance Registers (CANIDAR3)	R/W	25.3.2.16/25-28
0x28	MSCAN Identifier Mask Registers (CANIDMR0)	R/W	25.3.2.17/25-29
0x29	MSCAN Identifier Mask Registers (CANIDMR1)	R/W	25.3.2.17/25-29
0x2C	MSCAN Identifier Mask Registers (CANIDMR2)	R/W	25.3.2.17/25-29
0x2D	MSCAN Identifier Mask Registers (CANIDMR3)	R/W	25.3.2.17/25-29
0x30	MSCAN Identifier Acceptance Registers (CANIDAR4)	R/W	25.3.2.16/25-28
0x31	MSCAN Identifier Acceptance Registers (CANIDAR5)	R/W	25.3.2.16/25-28
0x34	MSCAN Identifier Acceptance Registers (CANIDAR6)	R/W	25.3.2.16/25-28
0x35	MSCAN Identifier Acceptance Registers (CANIDAR7)	R/W	25.3.2.16/25-28
0x38	MSCAN Identifier Mask Registers (CANIDMR4)	R/W	25.3.2.17/25-29
0x39	MSCAN Identifier Mask Registers (CANIDMR5)	R/W	25.3.2.17/25-29
0x3C	MSCAN Identifier Mask Registers (CANIDMR6)	R/W	25.3.2.17/25-29
0x3D	MSCAN Identifier Mask Registers (CANIDMR7)	R/W	25.3.2.17/25-29
0x40–0x5F	MSCAN Receive Message Buffer (CANRXFG)	R ²	25.4.3/25-44
0x60–0x7F	MSCAN Transmit Message Buffer (CANTXFG)	R/W ³	25.4.3/25-44

¹ Include short name and long name.

² Reserved bits and unused bits within the RX-buffer (CANRXFG) are read as x, because of RAM-based implementation.

³ Reserved bits and unused bits within the TX-buffer (CANTXFG) are read as x, because of RAM-based implementation.

25.3.1 Register Summary

Table 25-3 shows a summary of the registers.

Table 25-3. Register Summary

Name		7	6	5	4	3	2	1	0
0x00 CANCTL0	R	RXFRM	RXACT		SYNCH	TIME	WUPE	SLPRQ	INITRQ
	W								
0x01 CANCTL1	R	CANE	CLKSRC	LOOPB	LISTEN	0		SLPAK	INITAK
	W								
0x04 CANBTR0	R	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
	W								

Table 25-3. Register Summary (continued)

Name		7	6	5	4	3	2	1	0
0x05 CANBTR1	R	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
	W								
0x08 CANRFLG	R	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
	W								
0x09 CANRIER	R	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
	W								
0x0C CANTFLG	R	0	0	0	0	0	TXE2	TXE1	TXE0
	W								
0x0D CANTIER	R	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
	W								
0x10 CANTARQ	R	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
	W								
0x11 CANTAACK	R	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
	W								
0x14 CANTBSEL	R	0	0	0	0	0	TX2	TX1	TX0
	W								
0x15 CANIDAC	R	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
	W								
0x19 CANMISC	R	0	0	0	0	0	0	0	BOFFHOLD
	W								
0x1C CANRXERR	R	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
	W								
0x1D CANTXERR	R	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
	W								
0x20 CANIDAR0	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x21 CANIDAR1	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x24 CANIDAR2	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x25 CANIDAR3	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x28 CANIDMR0	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x29 CANIDMR1	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x2C CANIDMR2	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								

Table 25-3. Register Summary (continued)

Name		7	6	5	4	3	2	1	0
0x2D CANIDMR3	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x30 CANIDAR4	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x31 CANIDAR5	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x34 CANIDAR6	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x35 CANIDAR7	R	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	W								
0x38 CANIDMR4	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x39 CANIDMR5	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x3C CANIDMR6	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x3D CANIDMR7	R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	W								
0x40-0x5F CANRXFG	R	FOREGROUND RECEIVE BUFFER							
	W								
0x60-7F CANTXFG	R	FOREGROUND TRANSMIT BUFFER							
	W								

25.3.2 Register Descriptions

This section describes in detail all the registers and register bits in the MSCAN module. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagram, in bit order. All bits of all registers in this module are completely synchronous to internal clocks during a register read.

25.3.2.1 MSCAN Control 0 Register (CANCTL0)

This register provides for various control of the MSCAN module as described below.

Read: Anytime

Write: Anytime when out of initialization mode; exceptions are read-only RXACT and SYNCH, RXFRM (which is set by the module only), and INITRQ (which is also writable in initialization mode).

NOTE

The CANCTL0 register (except the WUPE, INTRQ, and SLPRQ bits) is held in the reset state when initialization mode is active (INTRQ=1 and INITAK=1). The CSWAI, TIME, WUPE, and SLPRQ bits are writable as soon as the initialization mode is complete (INTRQ=0 and INITAK=0).

MSCAN_BASE + 0x00

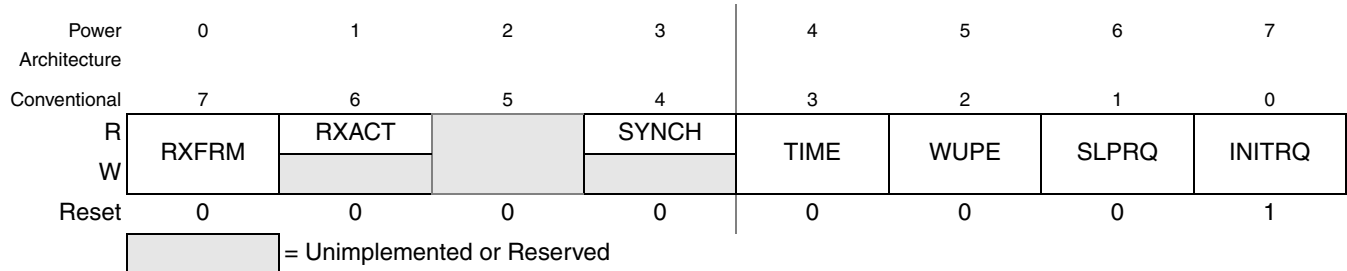


Figure 25-3. MSCAN Control 0 Register (CANCTL0)

Table 25-4. CANCTL0 Field Descriptions

Field	Description
RXFRM ¹	Received Frame Flag bit is read and clear only. It is set when a receiver has received a valid message correctly, independently of the filter configuration. After set, it remains set until cleared by software or reset. Clear by writing 1 to the bit. This bit is not valid in loop-back mode. when read: 0 No valid message was received since last clearing this flag. 1 A valid message was received since last clearing of this flag. when write: 0 no action 1 clear this bit
RXACT	Receiver Active Status bit indicates MSCAN is receiving a message. The receiver front end controls the flag. This bit is not valid in loop-back mode. 0 MSCAN is transmitting or idle ² . 1 MSCAN is receiving a message (including when arbitration is lost).
SYNCH	Synchronized Status bit indicates whether MSCAN is synchronized to the CAN bus and can participate in the communication process. It is set and cleared by MSCAN. 0 MSCAN is not synchronized to the CAN bus. 1 MSCAN is synchronized to the CAN bus.
TIME	Timer Enable bit activates an internal 16-bit wide free running timer clocked by the bit-clock. If timer is enabled, a 16-bit time stamp is assigned to each transmitted/received message within the active Tx/Rx buffer. As soon as a message is acknowledged on CAN, the time stamp is written to the highest bytes (0x1C, 0x1D) in the appropriate buffer (see Section 25.3.3, "Programmer's Model of Message Storage"). The internal timer is reset (all bits set to 0) when initialization mode is active. 0 Disable internal MSCAN timer. 1 Enable internal MSCAN timer.
WUPE ³	Wake-Up Enable bit lets MSCAN restart when being locked in idle state during sleep mode and traffic on CAN is detected.(see Section 25.4.8.1, "MSCAN Sleep Mode") 0 Wake-Up disabled. The MSCAN ignores traffic on CAN. 1 Wake-Up enabled.The MSCAN is able to restart.

Table 25-4. CANCTL0 Field Descriptions (continued)

Field	Description
SLPRQ ⁴	<p>Sleep Mode Request bit requests MSCAN enter sleep mode, an internal power saving mode (see Section 25.4.8.1, “MSCAN Sleep Mode”). The sleep mode request is serviced when the CAN bus is idle, i.e., the module is not receiving a message and all transmit buffers are empty. The module indicates entry to sleep mode by setting SLPAK = 1 (see Section 25.3.2.2, “MSCAN Control 1 Register (CANCTL1)”). Sleep mode is active until SLPRQ is cleared by the Power Architecture or, depending on the setting of WUPE, the MSCAN detects activity on the CAN bus and clears SLPRQ itself.</p> <p>0 Running. The MSCAN functions normally. 1 Sleep Mode Request. The MSCAN locks in idle state.</p>
INITRQ ^{5,6}	<p>Initialization Mode Request. When this bit is set by the Power Architecture, the MSCAN skips to initialization mode (see Section 25.4.8.2, “MSCAN Initialization Mode”). Any ongoing transmission or reception is aborted and synchronization to the CAN bus is lost. The module indicates entry to initialization mode by setting INITAK = 1 (Section 25.3.2.2, “MSCAN Control 1 Register (CANCTL1)”).</p> <p>The following registers enter their hard reset state and restore their default values: CANCTL0⁷, CANRFLG⁸, CANRIER⁹, CANTFLG, CANTIER, CANTARQ, CANTAAR, and CANTBSEL.</p> <p>The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0-7, and CANIDMR0-7 can only be written by the Power Architecture when the MSCAN is in initialization mode (INITRQ = 1 and INITAK = 1). The values of the error counters are not affected by initialization mode.</p> <p>When this bit is cleared by the Power Architecture, the MSCAN restarts and then tries to synchronize to the CAN bus. If the MSCAN is not in bus-off state, it synchronizes after 11 consecutive recessive bits on the CAN bus; if the MSCAN is in bus-off state, it continues to wait for 128 occurrences of 11 consecutive recessive bits.</p> <p>Writing to other bits in CANCTL0, CANRFLG, CANRIER, CANTFLG, or CANTIER must be done only after initialization mode is exited, which is INITRQ = 0 and INITAK = 0.</p> <p>0 Normal operation. 1 MSCAN in initialization state.</p>

¹ The MSCAN must be in normal mode for this bit to become set.

² See the Bosch CAN 2.0A/B specification for a detailed definition of transmitter and receiver states.

³ The Power Architecture has to make sure that the WUPE register and the WUPE wake-up interrupt enable register (see [Section 25.3.2.6, “MSCAN Receiver Interrupt Enable Register \(CANRIER\)”](#)) is enabled, if the recovery mechanism from deep sleep mode is required.

⁴ The Power Architecture cannot clear SLPRQ before the MSCAN has entered sleep mode (SLPRQ = 1 and SLPAK = 1).

⁵ The Power Architecture cannot clear INITRQ before the MSCAN has entered initialization mode (INITRQ = 1 and INITAK = 1).

⁶ To protect from accidentally violating the CAN protocol, the TXCAN pin is immediately forced to a recessive state when the initialization mode is requested by the Power Architecture. Thus, the recommended procedure is to bring the MSCAN into sleep mode (SLPRQ = 1 and SLPAK = 1) before requesting initialization mode.

⁷ Not including WUPE, INITRQ, and SLPRQ.

⁸ TSTAT1 and TSTAT0 are not affected by initialization mode.

⁹ RSTAT1 and RSTAT0 are not affected by initialization mode.

25.3.2.2 MSCAN Control 1 Register (CANCTL1)

This register provides for various control and handshake status information of the MSCAN module as described below.

Read: Anytime

Write: The CLKSRC, LOOPB, LISTEN, BORM, and WUPM bits can be written anytime when in initialization mode (INITRQ = 1 and INITAK = 1).

MSCAN_BASE + 0x001

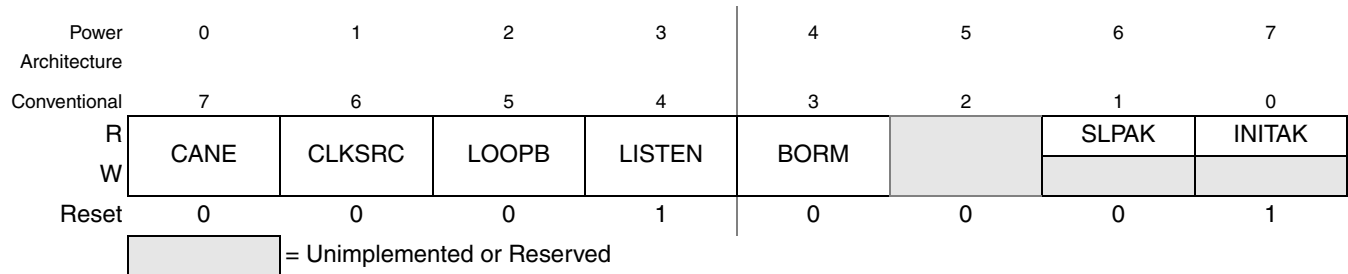


Figure 25-4. MSCAN Control 1 Register (CANCTL1)
(Register repeats for reference.)

Table 25-5. CANCTL1 Field Descriptions

Field	Description
CANE	MSCAN Enable 0 The MSCAN module is disabled. 1 The MSCAN module is enabled.
CLKSRC	MSCAN Clock Source. This bit defines the clock source for the MSCAN module (only for systems with a clock generation module; Section 25.4.5, “Clock System,” and Section Figure 25-39., “MSCAN Clocking Scheme,”). 0 The MSCAN clock source is the bus clock which may be originated from one of four sources: Section 5.2.5, “MSCAN Clock Generation.” 1 The MSCAN clock source is the ips clock.
LOOPB	Loop Back Self Test Mode. When bit is set, MSCAN does an internal loop-back that can be used for self test operation. Tx bit-stream output feeds back to receiver internally. RxCAN input pin is ignored and TxCAN output goes to recessive state (logic 1). MSCAN behaves as it does normally when transmitting and treats its own transmitted message as a message received from a remote node. In this state, MSCAN ignores bit sent during ACK slot in CAN frame acknowledge field to ensure proper reception of its own message. Both Tx and Rx interrupts are generated. 0 Loop Back Self Test disabled. 1 Loop Back Self Test enabled.
LISTEN	Listen Only Mode. This bit configures the MSCAN as a CAN bus monitor. When LISTEN is set, all valid CAN messages with matching ID are received, but no acknowledgement or error frames are sent out (see Section 25.4.7.5, “Listen-Only Mode.”). In addition, the error counters are frozen. Listen only mode supports applications which require hot plugging or throughput analysis. The MSCAN is unable to transmit any messages when listen only mode is active. 0 Normal operation. 1 Listen Only Mode activated.
BORM	Bus-Off Recovery Mode. This bits configures the bus-off state recovery mode of the MSCAN. Refer to Section 25.4.15, “Bus-Off Recovery,” for details. 1 = Bus-Off recovery upon request 0 = Automatic bus-off recovery

MSCAN_BASE + 0x001

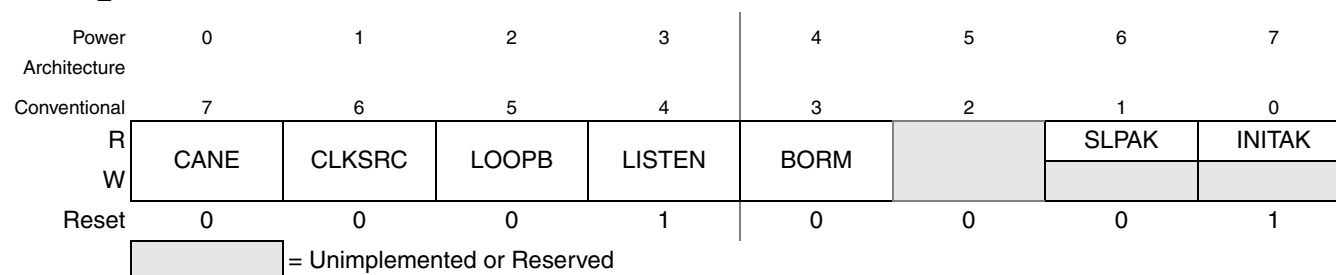


Figure 25-4. MSCAN Control 1 Register (CANCTL1)
(Register repeats for reference.)

Table 25-5. CANCTL1 Field Descriptions (continued)

Field	Description
SLPAK	<p>Sleep Mode Acknowledge. This flag indicates whether the MSCAN module has entered sleep mode (see Section 25.4.8.1, “MSCAN Sleep Mode”). It is used as a handshake flag for the SLPRQ sleep mode request. Sleep mode is active when SLPRQ = 1 and SLPAK = 1. Depending on the setting of WUPE, the MSCAN clears the flag if it detects activity on the CAN bus while in sleep mode.</p> <p>0 Running. The MSCAN operates normally. 1 Sleep Mode Active. The MSCAN has entered sleep mode.</p>
INITAK	<p>Initialization Mode Acknowledge. This flag indicates whether the MSCAN module is in initialization mode (see Section 25.4.8.2, “MSCAN Initialization Mode”). It is used as a handshake flag for the INITRQ initialization mode request. Initialization mode is active when INITRQ = 1 and INITAK = 1. The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0–CANIDAR7, and CANIDMR0–CANIDMR7 can be written only by the Power Architecture when the MSCAN is in initialization mode.</p> <p>0 Running. The MSCAN operates normally. 1 Initialization Mode Active. The MSCAN has entered initialization mode.</p>

25.3.2.3 MSCAN Bus Timing Register 0 (CANBTR0)

This register provides for various bus timing control of the MSCAN module as described below.

Read: Anytime

Write: Anytime in initialization mode (INITRQ=1 and INITAK=1)

MSCAN_BASE + 0x04

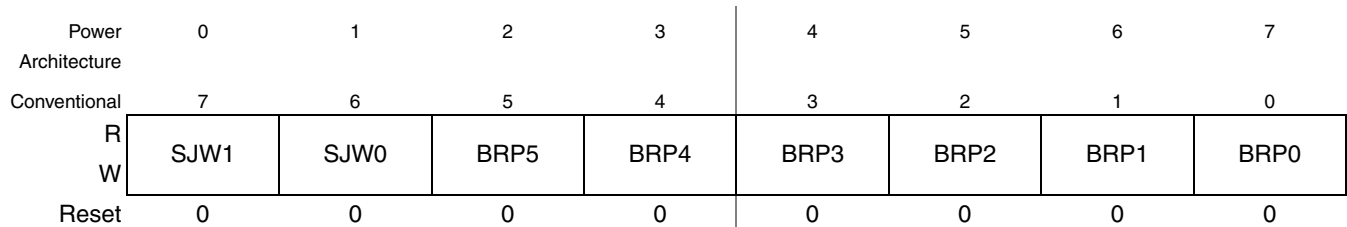


Figure 25-5. MSCAN Bus Timing Register 0 (CANBTR0)

Table 25-6. CANBTR0 Field Descriptions

Field	Description
SJW[1:0]	Synchronization Jump Width defines the maximum number of time quanta (Tq) clock cycles (refer to Figure 25-39 for Tq clock defination) a bit can be shortened or lengthened to achieve re-synchronization to data transitions on the bus. 00 1 Tq clock cycle 01 2 Tq clock cycles 10 3 Tq clock cycles 11 4 Tq clock cycles
BRP[5:0]	Baud Rate Prescaler bits determine time quanta (Tq) clock used to build up individual bit timing 000000 1 000001 2 000010 3 000011 4 111110 63 111111 64

25.3.2.4 MSCAN Bus Timing Register 1 (CANBTR1)

This register provides for various bus timing control of the MSCAN module as described below.

Read: Anytime

Write: Anytime in initialization mode (INITRQ=1 and INITAK=1)

MSCAN_BASE + 0x05

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R								
W	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
Reset	0	0	0	0	0	0	0	0

Figure 25-6. MSCAN Bus Timing Register 1 (CANBTR1)

Table 25-7. CANBTR1 Field Descriptions

Field	Description
SAMP	Sampling. This bit determines the number of CAN bus samples taken per bit time. (refer to Section Figure 25-40., “Segments within the Bit Time”) 0 One sample per bit. 1 Three samples per bit. If SAMP = 0, the resulting bit value is equal to the value of the single bit positioned at the sample point. If SAMP = 1, the resulting bit value is determined by using majority rule on the three total samples. For higher bit rates, it is recommended that only one sample is taken per bit time (SAMP = 0).
TSEG2	Time Segment 2. Time segments within the bit-time fix the number of clock cycles per bit-time and location of the sample point. (Section Figure 25-40., “Segments within the Bit Time”) 000 1 Tq clock cycle 001 2 Tq clock cycles 010 3 Tq clock cycles 111 8 Tq clock cycles
TSEG1	Time Segment 1. Time segments within the bit-time fix the number of clock cycles per bit-time and location of the sample point. (Section Figure 25-40., “Segments within the Bit Time”) 0000 1 Tq clock cycle 0001 2 Tq clock cycles 0010 3 Tq clock cycles 0011 4 Tq clock cycles 1110 15 Tq clock cycles 1111 16 Tq clock cycles

Bit time is determined by:

- Oscillator frequency
- Baud rate prescaler
- Number of time quanta (Tq) clock cycles per bit

$$\text{Bit Time} = \frac{(\text{Prescaler value})}{f_{\text{CANCLK}}} \cdot (1 + \text{TimeSegment1} + \text{Timesegment2})$$

25.3.2.5 MSCAN Receiver Flag Register (CANRFLG)

A flag can only be cleared when the condition that caused the setting is no longer valid and can only be cleared by software (writing a 1 to the corresponding bit position). Every flag has an associated interrupt enable bit in the CANRIER register.

The CANRFLG register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the initialization mode is left (INITRQ=0 and INITAK=0).

Read: Anytime

Write: Anytime when out of initialization mode, except RSTAT[1:0] and TSTAT[1:0] flags which are read-only; write of 1 clears flag; write of 0 ignored

NOTE

The CANRFLG register is held in the reset state¹ when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

1. The RSTAT[1:0], TSTAT[1:0] bits are not affected by initialization mode.

MSCAN_BASE + 0x08

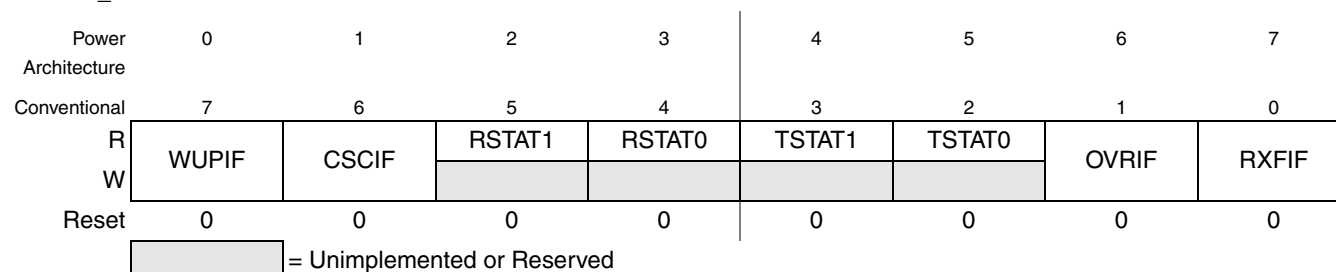
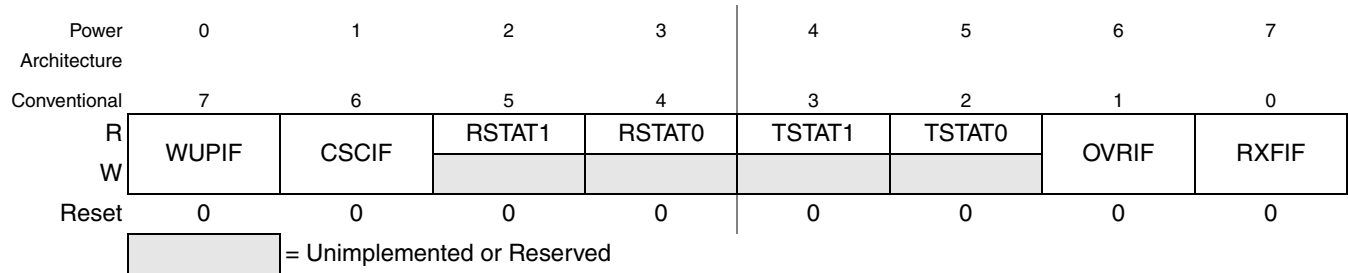


Figure 25-7. MSCAN Receiver Flag Register (CANRFLG)
(Register repeats for reference.)

Table 25-8. CANRFLG Field Descriptions

Field	Description
WUPIF	<p>Wake-Up Interrupt Flag. If the MSCAN detects CAN bus activity while in sleep mode (see Section 25.4.8.1, “MSCAN Sleep Mode,”) and WUPE = 1 in CANTCTL0 (see Section 25.3.2.1, “MSCAN Control 0 Register (CANCTL0)”), the module sets WUPIF. If not masked, a wake-up interrupt is pending while this flag is set.</p> <p>when read:</p> <p>0 No wake-up activity observed while in Sleep Mode.</p> <p>1 MSCAN detected activity on the bus and requested wake-up.</p> <p>when write:</p> <p>0 no action</p> <p>1 clear corresponding interrupt flag</p>
CSCIF	<p>CAN Status Change Interrupt Flag. This flag is set when the MSCAN changes its current CAN bus status due to the actual value of the transmit error counter (TEC) and the receive error counter (REC). An additional 4-bit (RSTAT[1:0], TSTAT[1:0]) status register, which is split into separate sections for TEC/REC, informs the system on the actual CAN bus status (see Section 25.3.2.6, “MSCAN Receiver Interrupt Enable Register (CANRIER)”). If not masked, an error interrupt is pending while this flag is set. CSCIF provides a blocking interrupt. That guarantees that the receiver/transmitter status bits (RSTAT/TSTAT) are only updated when no CAN status change interrupt is pending. If the TECs/RECs change their current value after the CSCIF is asserted, which would cause an additional state change in the RSTAT/TSTAT bits, these bits keep their status until the current CSCIF interrupt is cleared again.</p> <p>when read:</p> <p>0 No change in CAN bus status occurred since last interrupt</p> <p>1 MSCAN changed current CAN bus status</p> <p>when write:</p> <p>0 no action</p> <p>1 clear corresponding interrupt flag</p>
RSTAT[1:0]	<p>Receiver Status Bits. The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate receiver related CAN bus status of the MSCAN. The coding for the bits RSTAT1, RSTAT0 is:</p> <p>00 RxOK: $0 \leq \text{receive error counter} \leq 96$</p> <p>01 RxWRN: $96 < \text{receive error counter} \leq 127$</p> <p>10 RxERR: $127 < \text{receive error counter}$</p> <p>11 Bus-off¹: $\text{transmit error counter} > 255$</p>

MSCAN_BASE + 0x08



**Figure 25-7. MSCAN Receiver Flag Register (CANRFLG)
(Register repeats for reference.)**

Table 25-8. CANRFLG Field Descriptions (continued)

Field	Description
TSTAT[1:0]	Transmitter Status Bits. The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate transmitter related CAN bus status of the MSCAN. The coding for the bits TSTAT1, TSTAT0 is: 00 TxOK: 0 ≤ transmit error counter ≤ 96 01 TxWRN: 96 < transmit error counter ≤ 127 10 TxERR: 127 < transmit error counter ≤ 255 11 Bus-Off: transmit error counter > 255
OVRIF	Overrun Interrupt Flag is set when a data overrun condition occurs. If not masked, an error interrupt is pending while this flag is set. when read: 0 No data overrun condition. 1 A data overrun detected. when write: 0 no action 1 clear corresponding interrupt flag
RXFIF	Receive Buffer Full Flag is set by MSCAN when a new message is shifted into RX FIFO. Flag indicates whether the shifted buffer is loaded with a correctly received message (matching identifier, matching cyclic redundancy code (CRC) and no other errors detected). After Power Architecture reads message from RxFG buffer in Rx FIFO, RxF flag must be cleared to release the buffer. A set RxF flag prohibits shifting of next FIFO entry into foreground buffer (RxFG). If not masked, RX interrupt is pending while this flag is set. To ensure data integrity, do not read the Rx buffer registers while RXFIF flag is cleared. when read: 0 No new message available within the RxFG. 1 The receiver FIFO is not empty. A new message is available in the RxFG buffer. when write: 0 no action 1 clear corresponding interrupt flag

¹ Redundant Information for the most critical CAN bus status which is bus-off. This only occurs if the Tx error counter exceeds a number of 255 errors. Bus-off affects the receiver state. As soon as the transmitter leaves its bus-off state the receiver state skips to RxOK too. Refer also to TSTAT[1:0] coding in this register.

25.3.2.6 MSCAN Receiver Interrupt Enable Register (CANRIER)

This register contains the interrupt enable bits for the interrupt flags described in the CANRFLG register.

Read: Anytime

Write: Anytime when out of initialization mode

NOTE

WUPIE, CSCIE, OVRIE, and RXFIE are held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable when not in initialization mode (INITRQ=0 and INITAK=0).

MSCAN_BASE + 0x09

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-8. MSCAN Receiver Interrupt Enable Register (CANRIER)
(Register repeats for reference.)

Table 25-9. CANRIER Field Descriptions

Field	Description
WUPIE ¹	Wake-Up Interrupt Enable 0 No interrupt request is generated from this event. 1 A wake-up event causes a Wake-Up interrupt request.
CSCIE	CAN Status Change Interrupt Enable 0 No interrupt request is generated from this event. 1 A CAN Status Change event causes an error interrupt request.
RSTATE[1:0]	Receiver Status Change Enable. These RSTAT enable bits control the sensitivity level in which receiver state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level the RSTAT flags continue to indicate the actual receiver state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by receiver state changes. 01 Generate CSCIF interrupt only if the receiver enters or leaves bus-off ² state. Discard other receiver state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the receiver enters or leaves RxErr or Bus-Off state. Discard other receiver state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes

MSCAN_BASE + 0x09

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-8. MSCAN Receiver Interrupt Enable Register (CANRIER)
(Register repeats for reference.)

Table 25-9. CANRIER Field Descriptions (continued)

Field	Description
TSTATE[1:0]	Transmitter Status Change Enable. These TSTAT enable bits control the sensitivity level in which transmitter state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level, the TSTAT flags continue to indicate the actual transmitter state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by transmitter state changes. 00 Do not generate any CSCIF interrupt caused by transmitter state changes. 01 Generate CSCIF interrupt only if the transmitter enters or leaves Bus-Off state. Discard other transmitter state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the transmitter enters or leaves TxErr or Bus-Off state. Discard other transmitter state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes
OVRIE	Overrun Interrupt Enable 0 No interrupt request is generated from this event. 1 An overrun event causes an error interrupt request.
RXFIE	Receive Buffer Full Interrupt Enable 0 No interrupt request is generated from this event. 1 A receive buffer full (successful message reception) event causes a receiver interrupt request.

- ¹ WUPIE and WUPE (see [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\)”](#)) must both be enabled if the recovery mechanism from deep sleep mode is required.
- ² Bus-off state is defined by the CAN standard (see Bosch CAN 2.0A/B protocol specification: for only transmitters. Because the only possible state change for the transmitter from bus-off to TxOK also forces the receiver to skip its current state to RxOK, the coding of the RXSTAT[1:0] flags define an additional bus-off state for the receiver (see [Section 25.3.2.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#)).

25.3.2.7 MSCAN Transmitter Flag Register (CANTFLG)

The transmit buffer empty flags each have an associated interrupt enable bit in the CANTIER register.

Read: Anytime

Write: Anytime for TXEx flags when not in initialization mode; write of 1 clears flag, write of 0 ignored

NOTE

The CANTFLG register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the initialization mode is left (INITRQ=0 and INITAK=0).

MSCAN_BASE + 0x0C

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TXE2IF	TXE1IF	TXE0IF
W								
Reset	0	0	0	0	0	1	1	1


 = Unimplemented or Reserved

Figure 25-9. MSCAN Transmitter Flag Register (CANTFLG)

Table 25-10. CANTFLG Field Descriptions

Field	Description
TXE2IF, TXE1IF, TXE0IF	<p>Transmitter Buffer Empty. Those flags indicate that the associated transmit message buffer is empty, and thus not scheduled for transmission. The Power Architecture must clear the flag after a message is set up in the transmit buffer and is due for transmission. The MSCAN sets the flag after the message is sent successfully. The flag is also set by the MSCAN when the transmission request is successfully aborted due to a pending abort request (see Section 25.3.2.9, “MSCAN Transmitter Message Abort Request (CANTARQ)”). If not masked, a transmit interrupt is pending while this flag is set.</p> <p>Clearing a TXExIF flag also clears the corresponding ABTAKx (see Section 25.3.2.10, “MSCAN Transmitter Message Abort Acknowledge (CANTAACK)”). When a TXExIF flag is set, the corresponding ABTRQx bit is cleared (see Section 25.3.2.9, “MSCAN Transmitter Message Abort Request (CANTARQ)”).</p> <p>When listen-mode is active (see Section 25.3.2.2, “MSCAN Control 1 Register (CANCTL1)”) the TXExIF flags cannot be cleared and no transmission is started.</p> <p>Read and write accesses to the transmit buffer are blocked if the corresponding TXExIF bit is cleared (TXExIF = 0) and the buffer is scheduled for transmission.</p> <p>when read:</p> <ul style="list-style-type: none"> 0 The associated message buffer is full (loaded with a message due for transmission). 1 The associated message buffer is empty (not scheduled). <p>when write:</p> <ul style="list-style-type: none"> 0 no action 1 clear corresponding interrupt flag

25.3.2.8 MSCAN Transmitter Interrupt Enable Register (CANTIER)

This register contains the interrupt enable bits for the transmit buffer empty interrupt flags.

Read: Anytime

Write: Anytime when not in initialization mode

NOTE

The CANTIER register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the initialization mode is left (INITRQ=0 and INITAK=0).

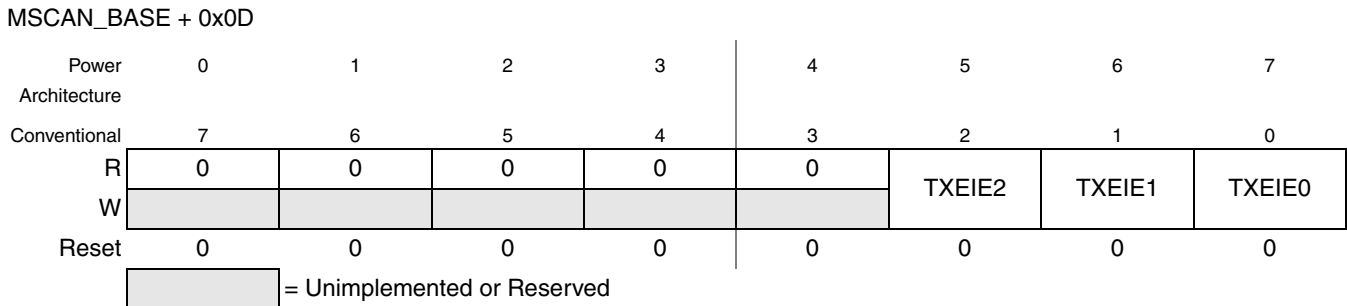


Figure 25-10. MSCAN Transmitter Interrupt Enable Register (CANTIER)

Table 25-11. CANTIER Field Descriptions

Field	Description
TXEIE[2:0]	Transmitter Empty Interrupt Enable 0 No interrupt request is generated from this event. 1 A transmitter empty (transmit buffer available for transmission) event causes a transmitter empty interrupt request.

25.3.2.9 MSCAN Transmitter Message Abort Request (CANTARQ)

This register provides for abort request of queued messages as described below.

Read: Anytime

Write: Anytime when not in initialization mode

NOTE

The CANTARQ register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable again as soon as the initialization mode is left (INITRQ=0 and INITAK=0).

MSCAN_BASE + 0x10

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
W								
Reset	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 25-11. MSCAN Transmitter Interrupt Enable Register (CANTIER)

Table 25-12. CANTIER Field Descriptions

Field	Description
ABTRQ[2:0]	<p>Abort Request. The Power Architecture sets the ABTRQx bit to request that a scheduled message buffer (TXExIF = 0) be aborted. The MSCAN grants the request if the message has not already started transmission, or if the transmission is not successful (lost arbitration or error). When a message is aborted, the associated TXExIF (see Section 25.3.2.7, “MSCAN Transmitter Flag Register (CANTFLG)”) and abort acknowledge flags (ABTAK, see Section 25.3.2.10, “MSCAN Transmitter Message Abort Acknowledge (CANTAACK)”) are set and a transmit interrupt occurs if enabled. The Power Architecture cannot reset ABTRQx. ABTRQx is reset whenever the associated TXExIF flag is set.</p> <p>0 No abort request. 1 Abort request pending.</p>

25.3.2.10 MSCAN Transmitter Message Abort Acknowledge (CANTAABK)

The CANTAABK register indicates the successful abort of a queued message if requested by the appropriate bits in the CANTARQ register.

Read: Anytime

Write: Unimplemented for ABTAKx flags;

NOTE

The CANTAABK register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1).

MSCAN_BASE + 0x11

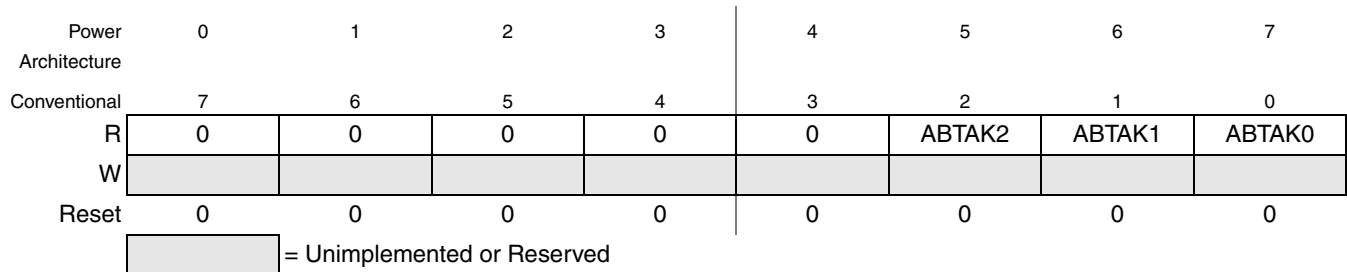


Figure 25-12. MSCAN Transmitter Message Abort Acknowledge (CANTAABK)

Table 25-13. CANTAABK Field Descriptions

Field	Description
ABTAK[2:0]	Abort Acknowledge flag acknowledges message was aborted due to pending Power Architecture abort request. After a specific message buffer is flagged empty, application software can use this flag to identify whether message was successfully aborted or was sent. Flag is cleared when the corresponding TxE flag is cleared. 0 The message was not aborted. 1 The message was aborted.

25.3.2.11 MSCAN Transmitter Buffer Selection (CANTBSEL)

This register allows the selection of the actual transmit message buffer, which is then accessible in the CANTXFG register space.

Read: Find the lowest ordered bit set to 1, all other bits are read as 0

Write: Anytime when not in initialization mode

NOTE

The CANTBSEL register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable again upon exiting the initialization mode (INITRQ=0 and INITAK=0).

The following gives a short programming example of usage of the CANTBSEL register:

To get the next available transmit buffer, application software must read the CANTFLG register and write this value back into the CANTBSEL register. In this example Tx buffers TX1 and TX2 are available. The value read from CANTFLG is therefore 0b0000_0110. When writing this value back to CANTBSEL, the Tx buffer TX1 is selected in the CANTXFG because the lowest numbered bit set to 1 is at bit position 1. Reading back this value out of CANTBSEL results in 0b0000_0010, because only the lowest numbered bit position set to 1 is presented. This mechanism eases the application software the selection of the next available Tx buffer.

- LDD CANTFLG; value read is 0b0000_0110
- STD CANTBSEL; value written is 0b0000_0110
- LDD CANTBSEL; value read is 0b0000_0010

If all transmit message buffers are deselected, no accesses are allowed to the CANTXFG registers.

MSCAN_BASE + 0x14

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TX2	TX1	TX0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 25-13. MSCAN Transmitter Buffer Selection (CANTBSEL)

Table 25-14. CANTBSEL Field Descriptions

Field	Description
TX[2:0]	<p>Transmit Buffer Select. The lowest numbered bit places the respective transmit buffer in the CANTXFG register space (e.g., TX1 = 1 and TX0 = 1 selects transmit buffer TX0; TX1 = 1 and TX0 = 0 selects transmit buffer TX1). Read and write accesses to the selected transmit buffer are blocked if the corresponding TXEx bit is cleared and the buffer is scheduled for transmission (see Section 25.3.2.7, “MSCAN Transmitter Flag Register (CANTFLG)”).</p> <p>0 The associated message buffer is deselected.</p> <p>1 The associated message Buffer is selected, if lowest numbered bit.</p>

25.3.2.12 MSCAN Identifier Acceptance Control Register (CANIDAC)

This register provides for identifier acceptance control as described below.

Read: Anytime

Write: Anytime in initialization mode (INITRQ=1 and INITAK=1), except bits IDHITx which are read-only

The IDHITx indicators are always related to the message in the foreground buffer (RxFG). When a message gets shifted into the foreground buffer of the receiver FIFO, the indicators are updated as well.

MSCAN_BASE + 0x15

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
W								
Reset	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 25-14. MSCAN Identifier Acceptance Control Register (CANIDAC)

Table 25-15. CANIDAC Field Descriptions

Field	Description
IDAM[1:0]	Identifier Acceptance Mode. Power Architecture sets these flags to define the identifier acceptance filter organization(see Section 25.4.3, "Identifier Acceptance Filter"). In filter closed mode, no message is accepted so the foreground buffer is never reloaded. 00 Two 32 bit Acceptance Filters 01 Four 16 bit Acceptance Filters 10 Eight 8 bit Acceptance Filters 11 Filter Closed
IDHIT[2:0]	Identifier Acceptance Hit Indicator. MSCAN sets these flags to indicate an identifier acceptance hit. (see Section 25.4.3, "Identifier Acceptance Filter") 000 Filter 0 Hit 001 Filter1 Hit 010 Filter 2 Hit 011 Filter 3 Hit 100 Filter 4 Hit 101 Filter 5 Hit 110 Filter 6 Hit 111 Filter 7 Hit

25.3.2.13 MSCAN MISC Register (CANMISC)

Read: Anytime

Write: Anytime; write of 1 clears flag; write of 0 ignored

MSCAN_BASE + 0x019

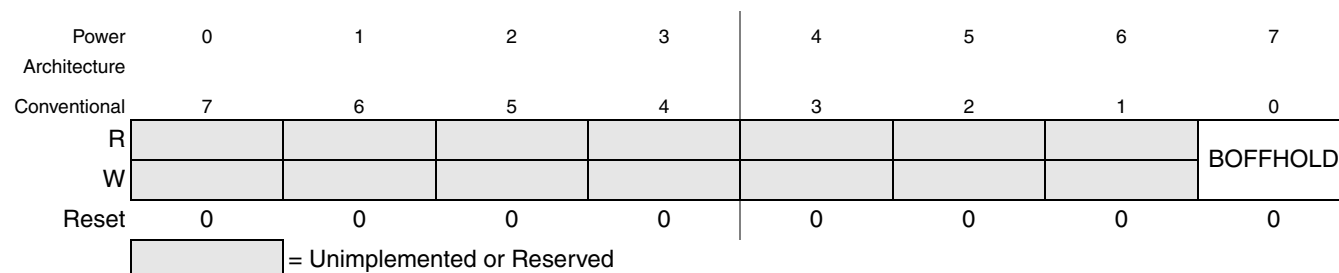


Figure 25-15. MSCAN MISC Register (CANMISC)

Table 25-16. CANMISC Field Descriptions

Field	Description
BOFFHOLD	<p>Bus-off State Hold Until User Request If BORM is set in MSCAN Control Register 1 (Section 25.3.2.2, “MSCAN Control 1 Register (CANCTL1)”), this bit indicates whether the module has entered the bus-off state. Clearing this bit requests the recovery from bus-off. Refer to 25.4.15, “Bus-Off Recovery,” for details.</p> <p>when read:</p> <p>0 Module is not bus-off or recovery has been requested by user in bus-off state</p> <p>1 Module is bus-off and holds this state until user request</p> <p>when write:</p> <p>0 no action</p> <p>1 clear corresponding interrupt flag</p>

25.3.2.14 MSCAN Receive Error Counter Register (CANRXERR)

This register reflects the status of the MSCAN receive error counter.

Read: Only when in sleep mode (SLPRQ=1 and SLPK=1) or initialization mode (INITRQ=1 and INITAK=1)

Write: Unimplemented

NOTE

Reading this register when in any other mode other than sleep or initialization mode, may return an incorrect value.

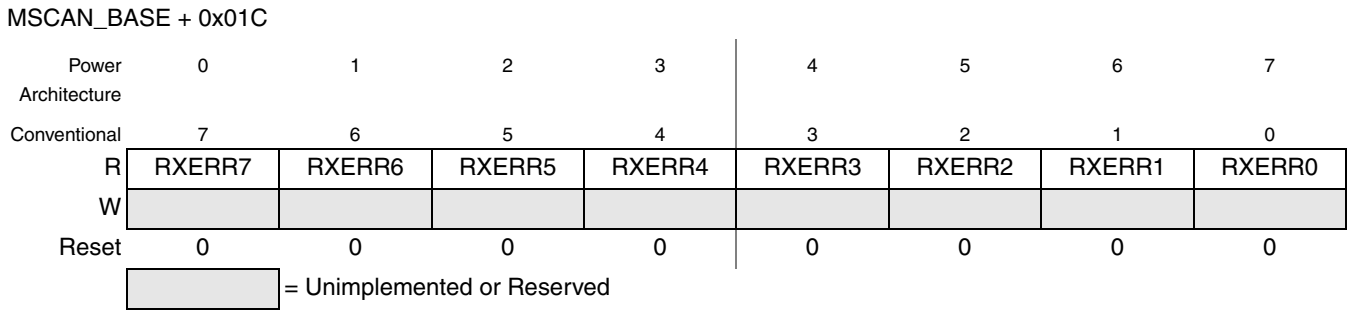


Figure 25-16. MSCAN Receive Error Counter Register (CANRXERR)

Table 25-17. CANRXERR Field Descriptions

Field	Description
RXERR[7:0]	This register reflects the status of the MSCAN receive error counter. Read: Only when in sleep mode (SLPRQ=1 and SLPK=1) or initialization mode (INITRQ=1 and INITAK=1)

25.3.2.15 MSCAN Transmit Error Counter Register (CANTXERR)

This register reflects the status of the MSCAN transmit error counter.

Read: Only when in sleep mode (SLPRQ=1 and SLPK=1) or initialization mode (INITRQ=1 and INITAK=1)

Write: Unimplemented

NOTE

Reading this register when in any other mode other than sleep or initialization mode, may return an incorrect value.

MSCAN_BASE + 0x01D

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
W								
Reset	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 25-17. MSCAN Transmit Error Counter Register (CANTXERR)

Table 25-18. CANTXERR Field Descriptions

Field	Description
TXERR[7:0]	This register reflects the status of the MSCAN receive error counter. Read: Only when in sleep mode (SLPRQ=1 and SLPK=1) or initialization mode (INITRQ=1 and INITAK=1)

25.3.2.16 MSCAN Identifier Acceptance Register (CANIDAR0–CANIDAR7)

On reception, each message is written into the background receive buffer. The Power Architecture is only signalled to read the message if it passes the criteria in the identifier acceptance and identifier mask registers (accepted); otherwise, the message is overwritten by the next message (dropped).

MSCAN_BASE + 0x20, 0x21, 0x24, 0x25

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R								
W	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset	0	0	0	0	0	0	0	0

Figure 25-18. MSCAN Identifier Acceptance Registers (1st Bank)

Offset CAN base address + 0x30, 0x31, 0x34, 0x35

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R								
W	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset	0	0	0	0	0	0	0	0

Figure 25-19. MSCAN Identifier Acceptance Registers (2nd Bank)

Table 25-19. CANIDAR0–CANIDAR7 Field Descriptions

Field	Description
AC[7:0]	Acceptance Code Bits. AC[7:0] comprises a user defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is masked with the corresponding identifier mask register.

The acceptance registers of the MSCAN are applied on the IDR0–IDR3 registers (see [Section 25.3.3.1, “Identifier Registers \(IDR0–IDR3\)”](#)) of incoming messages in a bit by bit manner (see [Section 25.4.3, “Identifier Acceptance Filter”](#)).

For extended identifiers, all four acceptance and mask registers are applied. For standard identifiers, only the first two (CANIDAR0/1, CANIDMR0/1) are applied.

25.3.2.17 MSCAN Identifier Mask Register (CANIDMR0–CANIDMR7)

MSCAN_BASE + 0x28, 0x29, 0x2C, 0x2D

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-20. MSCAN Identifier Mask Registers (1st Bank)

Offset CAN base address + 0x38, 0x39, 0x3C, 0x3D

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-21. MSCAN Identifier Mask Registers (2nd Bank)

Table 25-20. CANIDMR0–CANIDMR7 Field Descriptions

Field	Description
AM[7:0]	Acceptance Mask Bits. If a particular bit in this register is cleared, this indicates the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates the state of the corresponding bit in the identifier acceptance register does not affect whether or not message is accepted. 0 Match corresponding acceptance code register and identifier bits. 1 Ignore corresponding acceptance code register bit.

The identifier mask register specifies which of the corresponding bits in the identifier acceptance register are relevant for acceptance filtering.

- To receive standard identifiers in 32-bit filter mode, the last three bits (AM[0:2]) in the following mask registers must be programmed as don't care:
 - CANIDMR1
 - CANIDMR5
- To receive standard identifiers in 16-bit filter mode, the last three bits (AM[0:2]) in the following mask registers must be programmed as don't care:
 - CANIDMR1
 - CANIDMR3
 - CANIDMR5
 - CANIDMR7

25.3.3 Programmer's Model of Message Storage

The following section details the organization of the receive and transmit message buffers and the associated control registers.

Receive buffer start with base address 0x40, Transmit buffer start with base address 0x60

To simplify the programmer interface, the receive and transmit message buffers have the same outline. Each message buffer allocates 16 bytes in the memory map containing a 13 byte data structure.

An additional transmit buffer priority register (TBPR) is defined for the transmit buffers. Within the last two bytes of this memory map, the MSCAN stores a special 16-bit time stamp, which is sampled from an internal timer after successful transmission or reception of a message. This feature is only available for transmit and receiver buffers, if the TIME bit is set (see [Section 25.3.2.1, "MSCAN Control 0 Register \(CANCTL0\)"](#)).

The time stamp register is written by the MSCAN. The Power Architecture can only read these registers.

Table 25-21. Message Buffer Organization

Offset Address	Register
0x00	Identifier Register 0
0x01	Identifier Register 1
0x04	Identifier Register 2
0x05	Identifier Register 3
0x08	Data Segment Register 0
0x09	Data Segment Register 1
0x0C	Data Segment Register 2
0x0D	Data Segment Register 3
0x10	Data Segment Register 4
0x11	Data Segment Register 5
0x14	Data Segment Register 6
0x15	Data Segment Register 7
0x18	Data Length Register
0x19	Transmit Buffer Priority Register ¹
0x1C	Time Stamp Register (High Byte) ²
0x1D	Time Stamp Register (Low Byte) ³

¹ Not applicable for receive buffers

² Read-only for Power Architecture

³ Read-only for Power Architecture

[Figure 25-22](#) shows the common 13-byte data structure of receive and transmit buffers for extended identifiers. The mapping of standard identifiers into the IDR registers is shown in [Figure 25-23](#).

All bits of the receive and transmit buffers are x out of reset because of RAM-based implementation¹. All reserved or unused bits of the receive and transmit buffers always read 'x'.

1. Exception: The transmit priority registers are 0 out of reset.

MSCAN_BASE + (0X40 or 0x60)

Power Architecture		0	1	2	3	4	5	6	7
Conventional		7	6	5	4	3	2	1	0
Register Name									
0x00 IDR0	R W	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
0x01 IDR1	R W	ID20	ID19	ID18	SRR (=1)	IDE (=1)	ID17	ID16	ID15
0x04 IDR2	R W	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
0x05 IDR3	R W	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
0x08 DSR0	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x09 DSR1	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x0C DSR2	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x0D DSR3	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x10 DSR4	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x11 DSR5	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x14 DSR6	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x15 DSR7	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0x18 DLR	R W					DLC3	DLC2	DLC1	DLC0


 = Unused, always read 'x'

Figure 25-22. Receive/Transmit Message Buffer — Extended Identifier Mapping

Read: For transmit buffers, anytime when TXEx flag is set (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)). For receive buffers, only when RXF flag is set (see [Section 25.3.2.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#)).

Write: For transmit buffers, anytime when TXEx flag is set (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)). Unimplemented for receive buffers.

Reset: Undefined because of RAM-based implementation

MSCAN_BASE + (0X40 or 0x60)

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
Register Name								
IDR0 0x00	R	ID10 ID9 ID8 ID7 ID6 ID5 ID4 ID3						
IDR1 0x01	R	ID2 ID1 ID0 RTR IDE (=0)						
	W							

Figure 25-23. Receive/Transmit Message Buffer — Standard Identifier Mapping

25.3.3.1 Identifier Registers (IDR0–IDR3)

The identifier registers for an extended format identifier consist of a total of 32 bits; ID[28:0], SRR, IDE, and RTR bits. The identifier registers for a standard format identifier consist of a total of 13 bits; ID[10:0], RTR, and IDE bits.

25.3.3.1.1 IDR0–IDR3 for Extended Identifier Mapping

MSCAN_BASE + (0X40 or 0x60) + 0x00

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID28 ID27 ID26 ID25 ID24 ID23 ID22 ID21							
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-24. Identifier Register 0 (IDR0) — Extended Identifier Mapping

Table 25-22. IDR0 Register Field Descriptions — Extended

Field	Description
ID[28:21]	Extended Format Identifier. The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

MSCAN_BASE + (0X40 or 0x60) + 0x01

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID20	ID19	ID18	SRR (=1)	IDE (=1)	ID17	ID16	ID15
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-25. Identifier Register 1 (IDR1) — Extended Identifier Mapping

Table 25-23. IDR1 Register Field Descriptions — Extended

Field	Description
ID[20:18]	Extended Format Identifier. The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.
SRR	Substitute Remote Request. This fixed recessive bit is used only in extended format. It must be set to 1 by the user for transmission buffers and is stored as received on the CAN bus for receive buffers.
IDE	ID Extended. This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the Power Architecture how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)
ID[17:15]	Extended Format Identifier. The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

MSCAN_BASE + (0X40 or 0x60) + 0x04

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-26. Identifier Register 2 (IDR2) — Extended Identifier Mapping

Table 25-24. IDR2 Register Field Descriptions — Extended

Field	Description
ID[14:7]	Extended Format Identifier. The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

MSCAN_BASE + (0X40 or 0x60) + 0x05

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-27. Identifier Register 3 (IDR3) — Extended Identifier Mapping
Table 25-25. IDR3 Register Field Descriptions — Extended

Field	Description
ID[6:0]	Extended Format Identifier. The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.
RTR	Remote Transmission Request. This flag reflects the status of the remote transmission request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame

25.3.3.1.2 IDR0–IDR3 for Standard Identifier Mapping

MSCAN_BASE + (0X40 or 0x60) + 0x00

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-28. Identifier Register 0 — Standard Mapping

Table 25-26. IDR0 Register Field Descriptions — Standard

Field	Description
ID[10:3]	Standard Format Identifier. The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in Table 25-27 .

MSCAN_BASE + (0X40 or 0x60) + 0x01

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ID2	ID1	ID0	RTR	IDE (=0)			
W								
Reset:	x	x	x	x	x	x	x	x


 = Unused; always read 'x'

Figure 25-29. Identifier Register 1 — Standard Mapping

Table 25-27. IDR1 Register Field Descriptions

Field	Description
ID[2:0]	Standard Format Identifier. The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in Table 25-26 .
RTR	Remote Transmission Request. This flag reflects the status of the Remote Transmission Request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame
3 IDE	ID Extended. This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the Power Architecture how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)

25.3.3.2 Data Segment Registers (DSR0-7)

The eight data segment registers, each with bits DB[7:0], contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the corresponding DLR register.

MSCAN_BASE + (0x40 or 0x60)

+
 0x08 (DSR0)
 0x09 (DSR1)
 0x0C (DSR2)
 0x0D (DSR3)
 0x10 (DSR4)
 0x11 (DSR5)
 0x14 (DSR6)
 0x15 (DSR7)

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Reset:	x	x	x	x	x	x	x	x

Figure 25-30. Data Segment Registers (DSR0–DSR7) — Extended Identifier Mapping

Table 25-28. DSR0–DSR7 Register Field Descriptions

Field	Description
7:0 DB[7:0]	Data bits 7:0

25.3.3.3 Data Length Register (DLR)

This register keeps the data length field of the CAN frame.

MSCAN_BASE + (0x40 or 0x60) + 0x18

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R W					DLC3	DLC2	DLC1	DLC0
Reset:	x	x	x	x	x	x	x	x

= Unused; always read x

Figure 25-31. Data Length Register (DLR) — Extended Identifier Mapping

Table 25-29. DLR Register Field Descriptions

Field	Description
3:0 DLC[3:0]	Data Length Code Bits. The data length code contains the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0. The data byte count ranges from 0 to 8 for a data frame. Table 25-30 shows the effect of setting the DLC bits.

Table 25-30. Data Length Codes

Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

25.3.3.4 Transmit Buffer Priority Register (TBPR)

This register defines the local priority of the associated message buffer. The local priority is used for the internal prioritization process of the MSCAN and is defined to be highest for the smallest binary number. The MSCAN implements the following internal prioritization mechanisms:

- All transmission buffers with a cleared TXEx flag participate in the prioritization immediately before the SOF (start of frame) is sent.
- The transmission buffer with the lowest local priority field wins the prioritization.

In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins.

MSCAN_BASE + (0x40 or 0x60) + 0x19

Power Architecture	0	1	2	3	4	5	6	7
Convention	7	6	5	4	3	2	1	0
R	PRI07	PRI06	PRI05	PRI04	PRI03	PRI02	PRI01	PRI00
W								
Reset:	0	0	0	0	0	0	0	0

Figure 25-32. Transmit Buffer Priority Register (TBPR)

Read: Anytime when TXEx flag is set (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)).

Write: Anytime when TXEx flag is set (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)).

25.3.3.5 Time Stamp Register (TSRH–TSRL)

If the TIME bit is enabled, the MSCAN writes a special time stamp to the respective registers in the active transmit or receive buffer as soon as a message has been acknowledged on the CAN bus (see [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\)”](#)). The time stamp is written on the bit sample point for the recessive bit of the ACK delimiter in the CAN frame. In case of a transmission, the Power Architecture can only read the time stamp after the respective transmit buffer has been flagged empty.

The timer value, which is used for stamping, is taken from a free running internal CAN bit clock. A timer overrun is not indicated by the MSCAN. The timer is reset (all bits set to 0) during initialization mode. The Power Architecture can only read the time stamp registers.

MSCAN_BASE + (0x40 or 0x60) + 0x1C

Power Architecture	0	1	2	3	4	5	6	7
Convention	7	6	5	4	3	2	1	0
R	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-33. Time Stamp Register — High Byte (TSRH)

MSCAN_BASE + (0x40 or 0x60) + 0x1D

Power Architectur e	0	1	2	3	4	5	6	7
Convention al	7	6	5	4	3	2	1	0
R	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
W								
Reset:	x	x	x	x	x	x	x	x

Figure 25-34. Time Stamp Register — Low Byte (TSRL)

Read: Anytime when TXEx flag is set (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)).

Write: Unimplemented

25.4 Functional Description

25.4.1 General

This section provides a complete functional description of the MSCAN. It describes each of the features and modes listed in the introduction.

25.4.2 Message Storage

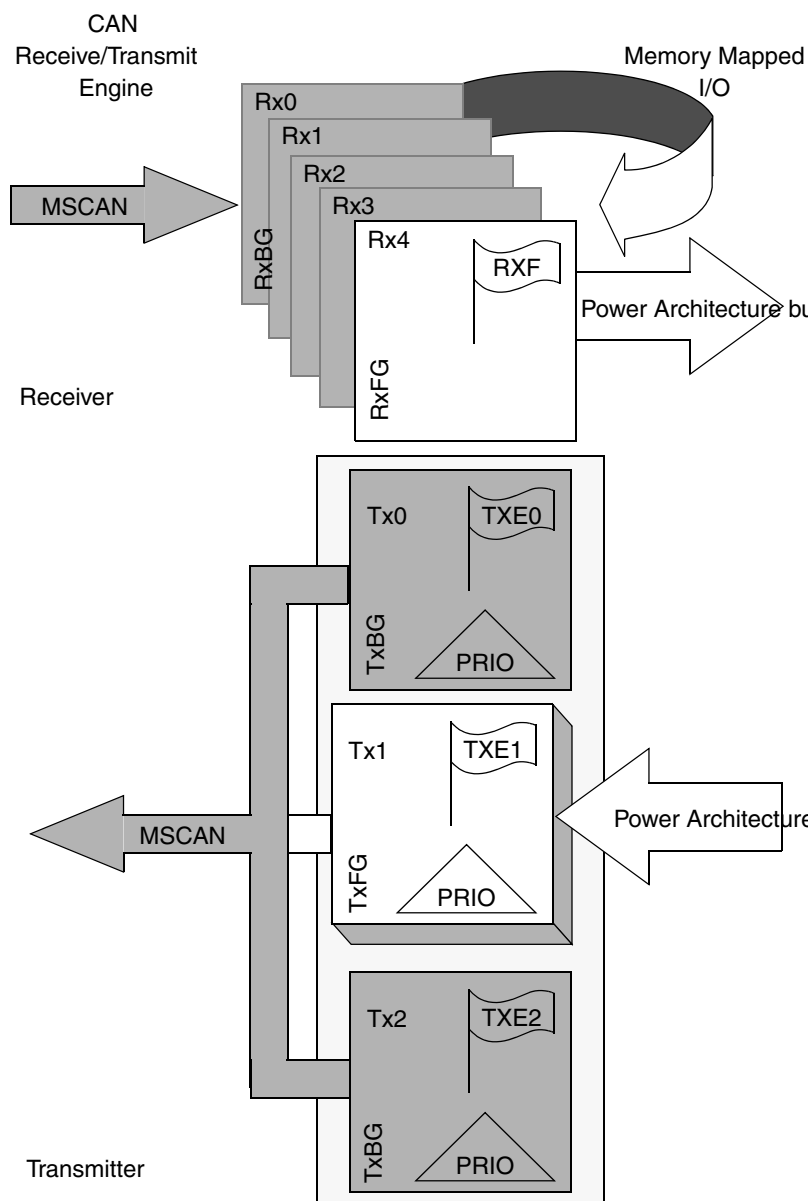


Figure 25-35. User Model for Message Buffer Organization

MSCAN facilitates a sophisticated message storage system which addresses the requirements of a broad range of network applications.

25.4.2.1 Message Transmit Background

Modern application layer software is built upon two fundamental assumptions:

- Any CAN node is able to send out a stream of scheduled messages without releasing the bus between the two messages. Such nodes arbitrate for the bus immediately after sending the previous message and only release the bus in case of lost arbitration.
- The internal message queue within any CAN node is organized so the highest priority message is sent out first if more than one message is ready to be sent.

The above behavior cannot be achieved with a single transmit buffer. That buffer must be reloaded right after the previous message is sent. This loading process lasts a finite amount of time and has to be completed within the inter-frame sequence (IFS)¹ to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires the Power Architecture react with short latencies to the transmit interrupt.

A double buffer scheme decouples the reloading of the transmit buffer from the actual message sending and reduces the reactivity requirements on the Power Architecture. Problems can arise if the sending of a message is finished while the Power Architecture reloads the second buffer. No buffer would be ready for transmission and the bus would be released.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The MSCAN has three transmit buffers.

The second requirement calls for some sort of internal prioritization which the MSCAN implements with the local priority concept described in [Section 25.4.2.2, “Transmit Structures”](#).

25.4.2.2 Transmit Structures

The MSCAN has a triple transmit buffer scheme which allows multiple messages to be set up in advance and achieve an optimized real-time performance. The three buffers are arranged as shown in [Figure 25-35](#).

All three buffers have a 13-byte data structure similar to the outline of the receive buffers (see [Section 25.3.3, “Programmer’s Model of Message Storage”](#)). An additional [Section 25.3.3.4, “Transmit Buffer Priority Register \(TBPR\)”](#) contains an 8-bit local priority field (PRIO). The remaining two bytes are used for time stamping of a message, if required (see [Section 25.3.3.5, “Time Stamp Register \(TSRH–TSRL\)”](#)).

To transmit a message, the Power Architecture must identify an available transmit buffer, which is indicated by a set transmitter buffer empty (TXEx) flag (see [Section 25.3.2.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)). If a transmit buffer is available, the Power Architecture must set a pointer to this buffer by writing to the CANTBSEL register (see [Section 25.3.2.11, “MSCAN Transmitter Buffer Selection \(CANTBSEL\)”](#)). This makes the respective buffer accessible within the CANTXFG address space (see [Section 25.3.3, “Programmer’s Model of Message Storage”](#)). The algorithmic feature associated with the CANTBSEL register simplifies the transmit buffer selection. In addition, this scheme makes the handler software simpler because only one address area is applicable for the transmit process, and the required address space is minimized.

1. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991.

The Power Architecture then stores the identifier, the control bits and the data content into one of the transmit buffers. Finally, the buffer is flagged as ready for transmission by clearing the associated TXE flag.

The MSCAN then schedules the message for transmission and signals the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt [Section 25.4.11.1, “Transmit Interrupt,”](#) is generated¹ when TXEx is set and can drive the application software to reload the buffer.

In case more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the MSCAN uses the local priority setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit local priority field (PRIO). The application software programs this field when the message is set up. The local priority reflects the priority of this particular message relative to the set of messages transmitted from this node. The lowest binary value of the PRIO field is defined to be the highest priority. The internal scheduling process takes place when the MSCAN arbitrates for the bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message in one of the three transmit buffers. Because messages that are already in transmission cannot be aborted, the user must request the abort by setting the corresponding abort request bit (ABTRQ) (see [Section 25.3.2.9, “MSCAN Transmitter Message Abort Request \(CANTARQ\)”](#).) The MSCAN then grants the request, if possible, by:

1. Setting the corresponding abort acknowledge flag (ABTAK) in the CANTAACK register.
2. Setting the associated TXE flag to release the buffer.
3. Generating a transmit interrupt. The transmit interrupt handler software can determine from the setting of the ABTAK flag whether the message was aborted (ABTAK = 1) or sent (ABTAK = 0).

25.4.2.3 Receive Structures

The received messages are stored in a five-stage input FIFO. The five message buffers are alternately mapped into a single memory area as seen in [Figure 25-35](#). While the background receive buffer (RxBG) is exclusively associated with the MSCAN, the foreground receive buffer (RxFG) is addressable by the Power Architecture as seen in [Figure 25-35](#). This scheme simplifies the manager software as only one address area is applicable for the receive process.

All receive buffers have a size of 15 bytes to store the CAN control bits, the identifier (standard or extended), the data contents and a time stamp, if enabled (for details [Section 25.4.3, “Identifier Acceptance Filter”](#))².

The receiver full flag (RXFIF) [Section 25.3.2.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#) signals the status of the foreground receive buffer. When the buffer contains a correctly received message with a matching identifier, this flag is set.

On reception, each message is checked to see if it passes the filter ([Section 25.3.2.12, “MSCAN Identifier Acceptance Control Register \(CANIDAC\),”](#)) and in parallel, is written into the active RxBG. After successful reception of a valid message the MSCAN shifts the content of RxBG into the receiver FIFO³,

1. The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.

2. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for details.

and generates a receive interrupt [Section 25.4.11.2, “Receive Interrupt”](#) to the Power Architecture¹ by set RXFIF. The receive manager has to read the received message from the RxFG, reset the RXFIF flag to acknowledge the interrupt, and release the foreground buffer. A new message, which can follow immediately after the IFS field of the CAN frame, is received into the next available RxBG. If the MSCAN receives an invalid message in its RxBG (wrong identifier, transmission errors etc.), the actual contents of the buffer are over-written by the next message. The buffer is not shifted into the FIFO.

When the MSCAN module is transmitting, the MSCAN receives its own transmitted messages into the background receive buffer (RxBG), but does not shift it into the receiver FIFO, generate a receive interrupt, or acknowledge its own messages on the CAN bus. The exception to this rule is in loop back mode [Section 25.3.2.2, “MSCAN Control 1 Register \(CANCTL1\),”](#) where the MSCAN treats its own messages exactly like all other incoming messages. The MSCAN receives its own transmitted messages in the event that it loses arbitration². If arbitration is lost, the MSCAN must be prepared to become a receiver.

An overrun condition occurs when all receive message buffers in the FIFO are filled with correctly received messages with accepted identifiers and another message is correctly received from the bus with an accepted identifier. The latter message is discarded and an error interrupt with overrun indication is generated if enabled [Section 25.4.11.4, “Error Interrupt”](#). The MSCAN remains able to transmit messages while the receiver FIFO is being filled, but all incoming messages are discarded. As soon as a receive buffer in the FIFO is available again, new valid messages are accepted.

25.4.3 Identifier Acceptance Filter

The MSCAN identifier acceptance registers ([Section 25.3.2.12, “MSCAN Identifier Acceptance Control Register \(CANIDAC\)”](#)) define the acceptable patterns of the standard or extended identifier (ID10 - ID0 or ID28 - ID0). Any of these bits can be marked don't care in the MSCAN identifier mask registers [Section 25.3.2.17, “MSCAN Identifier Mask Register \(CANIDMR0–CANIDMR7\)”](#).

A filter hit is indicated to the application software by a set receive buffer full flag (RXF=1) and three bits in the CANIDAC register [Section 25.3.2.16, “MSCAN Identifier Acceptance Register \(CANIDAR0–CANIDAR7\)”](#). These identifier hit flags (IDHIT2-0) clearly identify the filter section that caused the acceptance. They simplify the application software's task to identify the cause of the receiver interrupt. In case more than one hit occurs (two or more filters match), the lower hit has priority.

A flexible programmable generic identifier acceptance filter has been introduced to reduce the Power Architecture interrupt loading. The filter is programmable to operate in four different modes³:

- Two identifier acceptance filters, each to be applied to:
 - The full 29 bits of the extended identifier and to the following bits of the CAN 2.0B frame:
 - Remote transmission request (RTR)
 - Identifier extension (IDE)

3. Only if the RXF flag is not set.

1. The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.

2. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for details.

3. For a better understanding of references made within the filter mode description, reference the Bosch specification dated September 1991 which details the CAN 2.0A/B protocol.

- Substitute remote request (SRR)
- The 11 bits of the standard identifier plus the RTR and IDE bits of the CAN 2.0A/B messages¹. This mode implements two filters for a full length CAN 2.0B compliant extended identifier. [Figure 25-39](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces a filter 0 hit. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces a filter 1 hit.
- Four identifier acceptance filters, each to be applied to
 - a) the 14 most significant bits of the extended identifier plus the SRR and IDE bits of CAN 2.0B messages or
 - b) the 11 bits of the standard identifier, the RTR and IDE bits of CAN 2.0A/B messages. [Figure 25-40](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces filter 0 and 1 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 2 and 3 hits.
- Eight identifier acceptance filters, each to be applied to the first 8 bits of the identifier. This mode implements eight independent filters for the first 8 bits of a CAN 2.0A/B compliant standard identifier or a CAN 2.0B compliant extended identifier. [Figure 25-38](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces filter 0 to 3 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 4 to 7 hits.
- Closed filter. No CAN message is copied into the foreground buffer RxFG, and the RXF flag is never set.

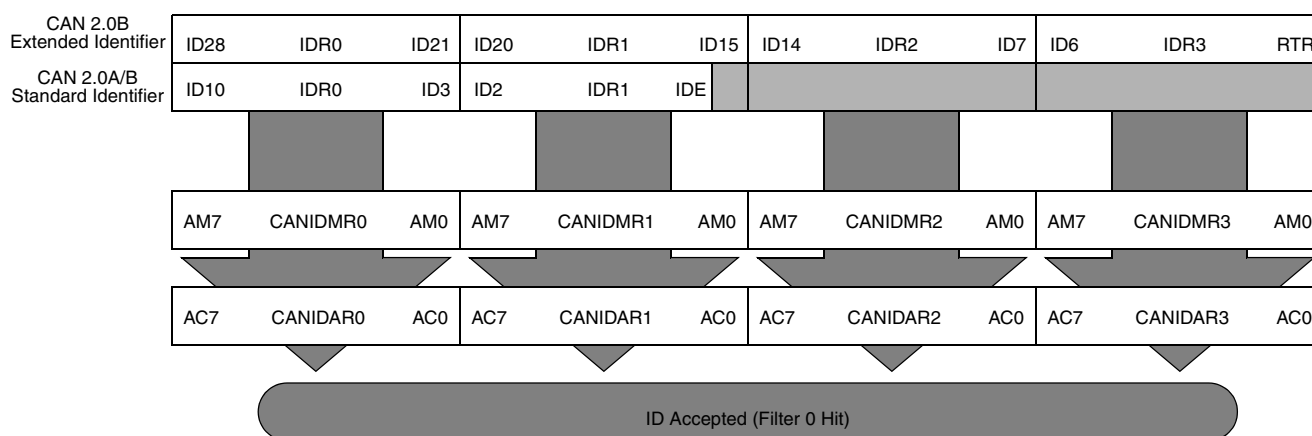


Figure 25-36. 32-Bit Maskable Identifier Acceptance Filter

¹ Although this mode can be used for standard identifiers, it is recommended to use the four or eight identifier acceptance filters for standard identifiers

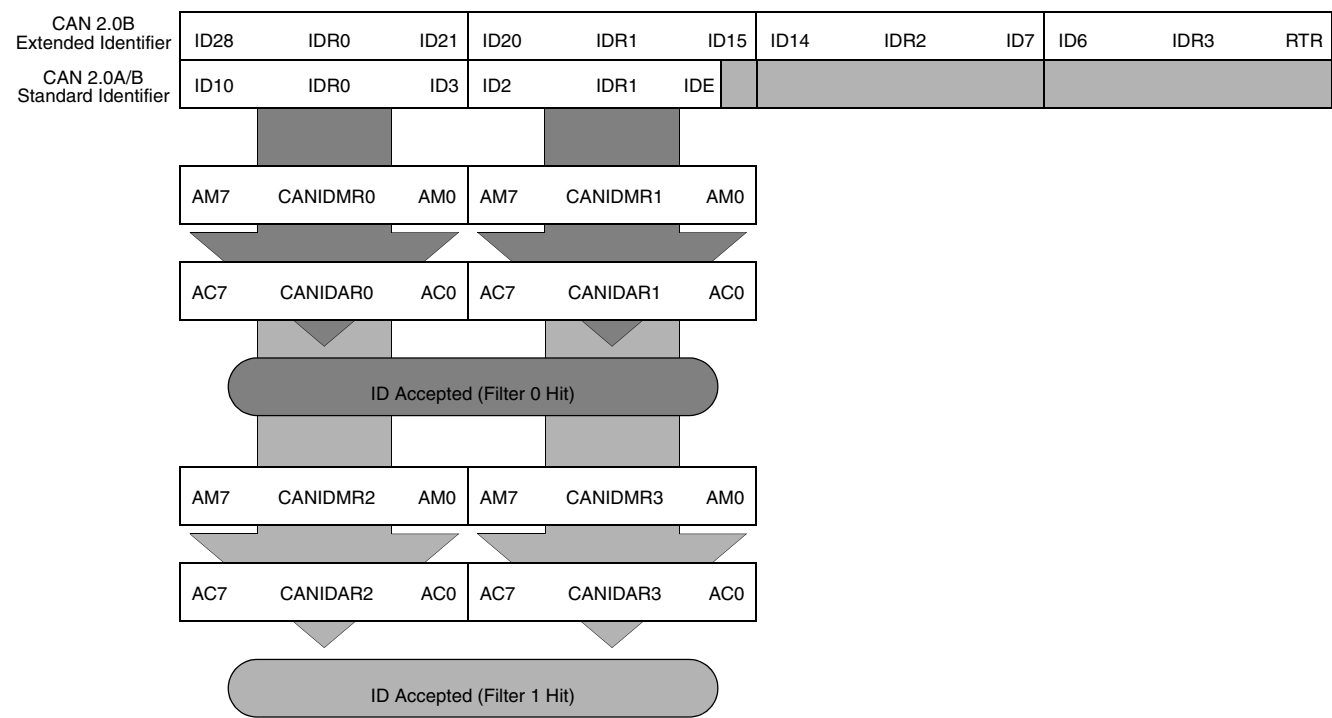


Figure 25-37. 16-Bit Maskable Identifier Acceptance Filters

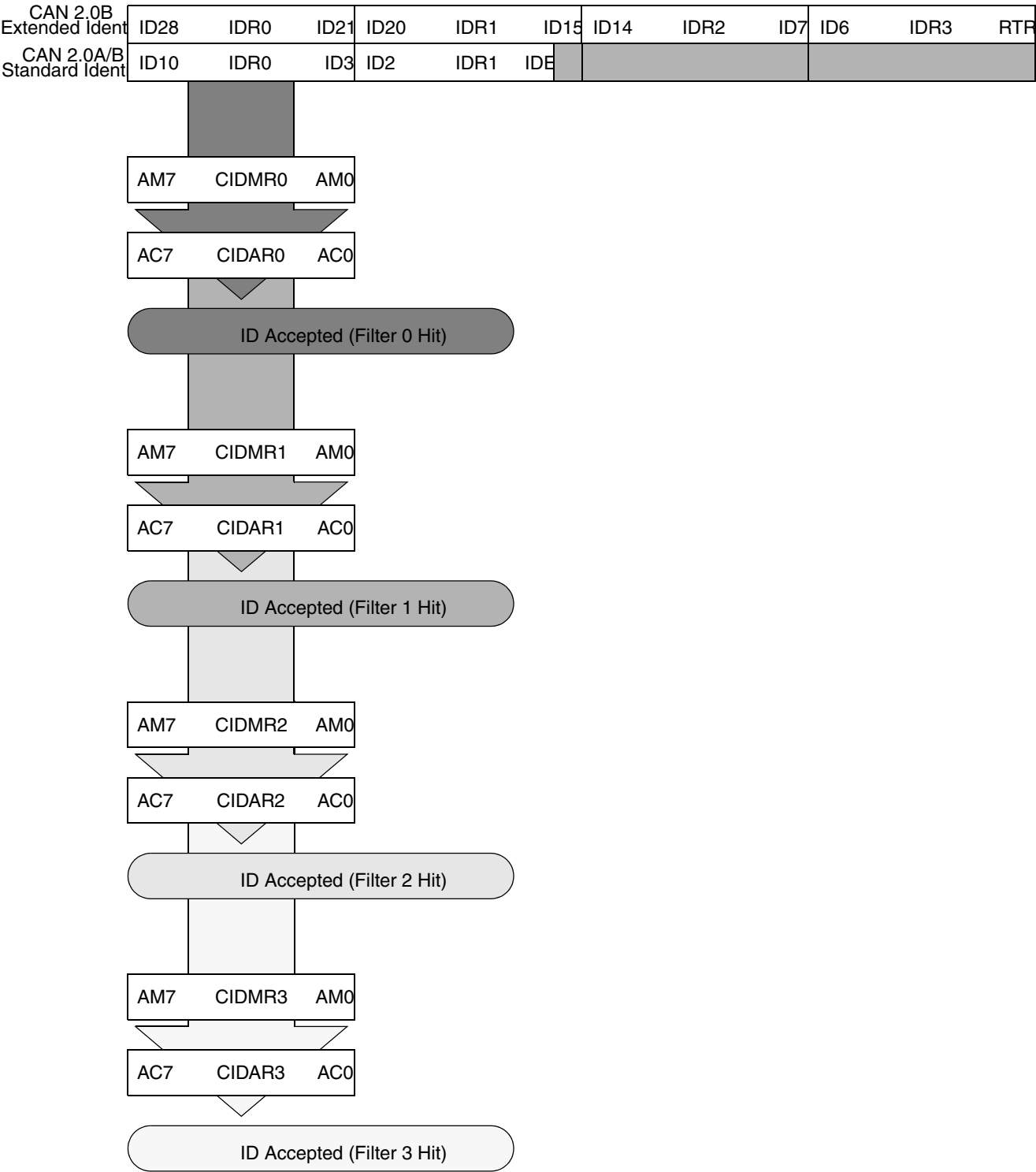


Figure 25-38. 8-Bit Maskable Identifier Acceptance Filters

25.4.4 Protocol Violation Protection

The MSCAN protects you from accidentally violating the CAN protocol through programming errors. The protection logic implements the following features:

- The receive and transmit error counters cannot be written or otherwise manipulated.
- All registers which control the configuration of the MSCAN cannot be modified while the MSCAN is on-line. The MSCAN has to be in Initialization Mode. The corresponding INITRQ/INITAK handshake bits in the CANCTL0/CANCTL1 registers [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\)”](#) serve as a lock to protect the following registers:
 - MSCAN Control 1 Register (CANCTL1)
 - MSCAN Bus Timing Registers 0 and 1 (CANBTR0, CANBTR1)
 - MSCAN Identifier Acceptance Control Register (CANIDAC)
 - MSCAN Identifier Acceptance Registers (CANIDAR0-7)
 - MSCAN Identifier Mask Registers (CANIDMR0-7)
- The TXCAN pin is immediately forced to a recessive state when the MSCAN goes into the power down mode or initialization mode (see [Section 25.4.8.3, “MSCAN Power Down Mode”](#) and [Section 25.4.8.2, “MSCAN Initialization Mode”](#)).

25.4.5 Clock System

[Figure 25-39](#) shows the structure of the MSCAN clock generation circuitry. With this flexible clocking scheme, the MSCAN can manage CAN bus rates ranging from 10 Kbps up to 1 Mbps.

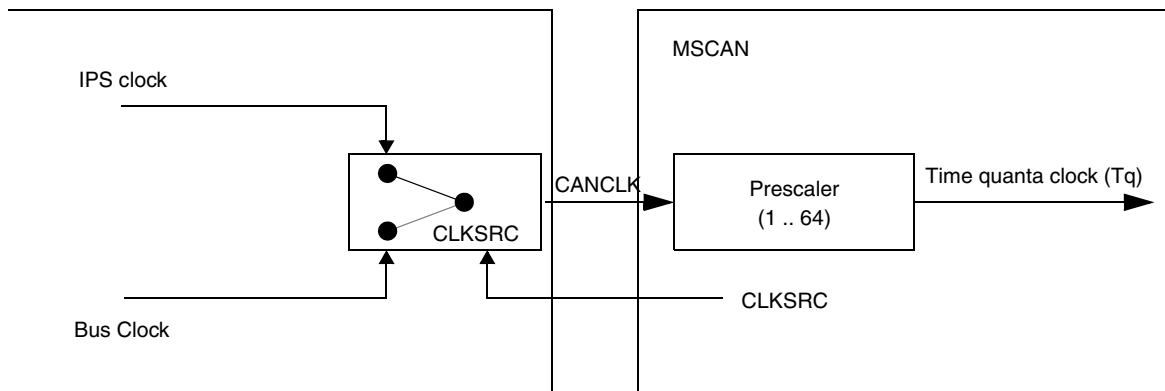


Figure 25-39. MSCAN Clocking Scheme

The clock source bit (CLKSRC) in the CANCTL1 register [Section 25.3.2.2, “MSCAN Control 1 Register \(CANCTL1\)”](#) defines whether the internal CANCLK is connected to the output of the system bus clock or to the IPS clock.

NOTE

Both MSCAN modules can have different selected clock sources. To select the bus clock, the CLKSRC bit in the CANCTL1 register must be set.

The clock source has to be chosen such that the tight oscillator tolerance requirements (up to 0.4%) of the CAN protocol are met. Additionally, for high CAN bus rates (1 Mbps), a 45% – 55% duty cycle of the clock is required.

A programmable prescaler generates the time quanta (Tq) clock from CANCLK. A time quantum is the atomic unit of time handled by the MSCAN.

Eqn. 25-2

$$f_{Tq} = \frac{f_{CANCLK}}{(\text{Prescaler} \cdot \text{value})}$$

A bit time is subdivided into three segments^{1 2} (reference [Figure 25-40](#)):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the PROP_SEG and the PHASE_SEG1 of the CAN standard.
- Time Segment 2: This segment represents the PHASE_SEG2 of the CAN standard. Setting the parameter TSEG2 to consist of 2 to 8 time quanta long can be programmed it.

Eqn. 25-3

$$\text{Bit P Rate} = \frac{f_{Tq}}{(\text{number P of P Time P Quanta})}$$

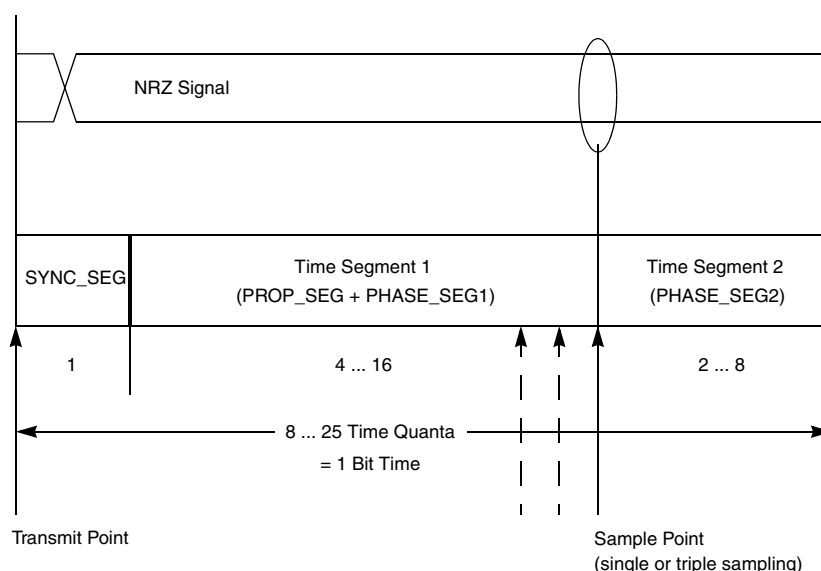


Figure 25-40. Segments within the Bit Time

Table 25-31. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.

1. For further explanation of the under-lying concepts, refer to ISO/DIS 11519-1, Section 10.3.

2. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 25-31. Time Segment Syntax (continued)

Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the bus at this point. If the three samples per bit option is selected, this point marks the position of the third sample.

The synchronization jump width¹ can be programmed in a range of 1 to 4 time quanta by setting the SJW parameter.

The above parameters are set by programming the MSCAN Bus Timing Registers (CANBTR0, CANBTR1) (see [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\)”](#) and [Section 25.3.2.4, “MSCAN Bus Timing Register 1 \(CANBTR1\)”](#)).

[Table 25-32](#) gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

Ensure the bit time settings are in compliance with the CAN standard.

Table 25-32. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	TSEG1	Time Segment 2	TSEG2	Synchronization Jump Width	SJW
5 .. 10	4 .. 9	2	1	1 .. 2	0 .. 1
4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
5 .. 12	4 .. 11	4	3	1 .. 4	0 .. 3
6 .. 13	5 .. 12	5	4	1 .. 4	0 .. 3
7 .. 14	6 .. 13	6	5	1 .. 4	0 .. 3
8 .. 15	7 .. 14	7	6	1 .. 4	0 .. 3
9 .. 16	8 .. 15	8	7	1 .. 4	0 .. 3

25.4.6 Timer Link

The MSCAN generates an internal time stamp when a valid frame is received or transmitted and the TIME bit is enabled. Because the CAN specification defines a frame to be valid if no errors occur before the End of Frame (EOF) field is transmitted successfully, the actual value of an internal timer is written at EOF to the appropriate time stamp position within the transmit buffer. For receive frames, the time stamp is written to the receive buffer.

25.4.7 Modes of Operation

25.4.7.1 Normal Mode

The MSCAN module behaves as described within this specification in all normal modes.

1. Reference the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

25.4.7.2 Initialization Mode

The MSCAN is put into initialization mode when INITRQ=1 and INITAK=0.

25.4.7.3 Sleep mode

The MSCAN is put into sleep mode When SLPRQ = 1 and SLPK = 1.

25.4.7.4 Power down mode

The MSCAN is put into power down mode when Power Architecture goes into deep sleep mode.

25.4.7.5 Listen-Only Mode

In an optional bus monitoring mode (listen-only), the CAN node can receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus. In addition, it cannot start a transmission. If the MAC sub-layer is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so the MAC sub-layer monitors this dominant bit; although, the CAN bus may remain in recessive state externally.

25.4.8 Low Power Options

If the MSCAN is disabled (CANE=0), the MSCAN clocks are stopped for power savings.

If the MSCAN is enabled (CANE=1), the MSCAN has two additional modes with reduced power consumption, compared to normal mode: sleep and power down mode. In sleep mode, power consumption is reduced by stopping all clocks except those to access the registers from the Power Architecture side. In power down mode, all clocks are stopped and no power is consumed.

[Table 25-33](#) summarizes the MSCAN modes. A particular combination of modes is entered by the given settings on the SLPRQ/SLPAK bits.

For all modes, an MSCAN wake-up interrupt can only occur if the MSCAN is in sleep mode (SLPRQ=1 and SLPK=1), wake-up functionality is enabled (WUPE=1), and the wake-up interrupt is enabled (WUPIE=1).

Table 25-33. MSCAN Operating Modes

MSCAN Mode			
Normal	Power Down(Power Architecture enter deep sleep)	Sleep	(CANE=0)
SLPRQ = 0 SLPAK = 0	SLPRQ = X SLPAK = X	SLPRQ = 1 SLPAK = 1	SLPRQ = X ¹ SLPAK = X

¹ X means don't care

25.4.8.1 MSCAN Sleep Mode

The Power Architecture can request the MSCAN to enter this low power mode by asserting the SLPRQ bit in the CANCTL0 register. The time when the MSCAN enters sleep mode depends on a fixed-synchronization delay and its current activity:

- If there are one or more message buffers scheduled for transmission (TXEx = 0), the MSCAN continues to transmit until all transmit message buffers are empty (TXEx = 1, transmitted successfully or aborted) and then goes into sleep mode.
- If the MSCAN is receiving, it continues to receive and goes into sleep mode as soon as the CAN bus next becomes idle.
- If the MSCAN is neither transmitting nor receiving, it immediately goes into sleep mode.

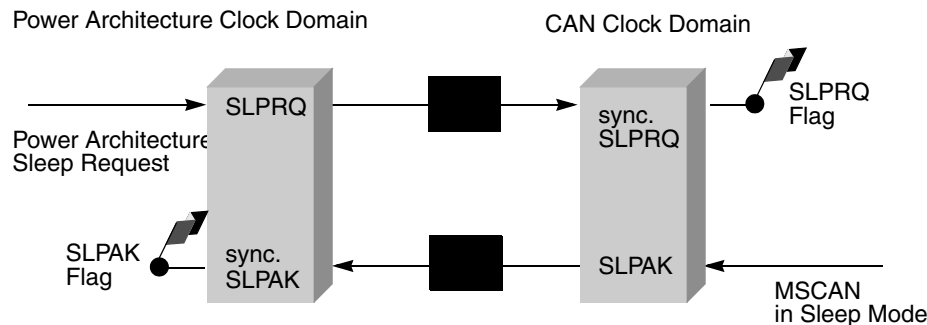


Figure 25-41. Sleep Request/Acknowledge Cycle

NOTE

The application software must avoid setting up a transmission (by clearing one or more TXEx flag[s]) and immediately request sleep mode (by setting SLPRQ). It depends on the exact sequence of operations whether the MSCAN starts transmitting or goes into sleep mode directly.

If sleep mode is active, the SLPRQ and SLPK bits are set (Figure 25-41). The application software must use SLPK as a handshake indication for the request (SLPRQ) to go into sleep mode.

When in sleep mode (SLPRQ = 1 and SLPK = 1), the MSCAN stops its internal clocks. However, clocks that allow register accesses from the Power Architecture side continue to run.

If the MSCAN is in bus-off state, it stops counting the 128 occurrences of 11 consecutive recessive bits due to the stopped clocks. The TXCAN pin remains in a recessive state. If RXF = 1, the message can be read and RXF can be cleared. Shifting a new message into the foreground buffer of the receiver FIFO (RxFG) does not take place while in sleep mode.

It is possible to access the transmit buffers and to clear the associated TXE flags. No message abort takes place while in sleep mode.

If the WUPE bit in CANCLT0 is not asserted, the MSCAN masks any activity it detects on CAN. The RXCAN pin is therefore held internally in a recessive state. This locks the MSCAN in sleep mode. The MSCAN is able to leave sleep mode (wake up) only when:

- CAN bus activity occurs and WUPE = 1
- or

- the Power Architecture clears the SLPRQ bit

NOTE

The Power Architecture cannot clear the SLPRQ bit before sleep mode (SLPRQ=1 and SLPAK=1) is active. After wake-up, the MSCAN waits for 11 consecutive recessive bits to synchronize to the bus. As a consequence, if the MSCAN is woken-up by a CAN frame, this frame is not received. The receive message buffers (RxFG and RxBG) contain messages if they were received before sleep mode was entered. All pending actions are executed upon wake-up; copying of RxBG into RxFG, message aborts and message transmissions. If the MSCAN remains in bus-off state after sleep mode was left, it continues counting the 128*11 consecutive recessive bits.

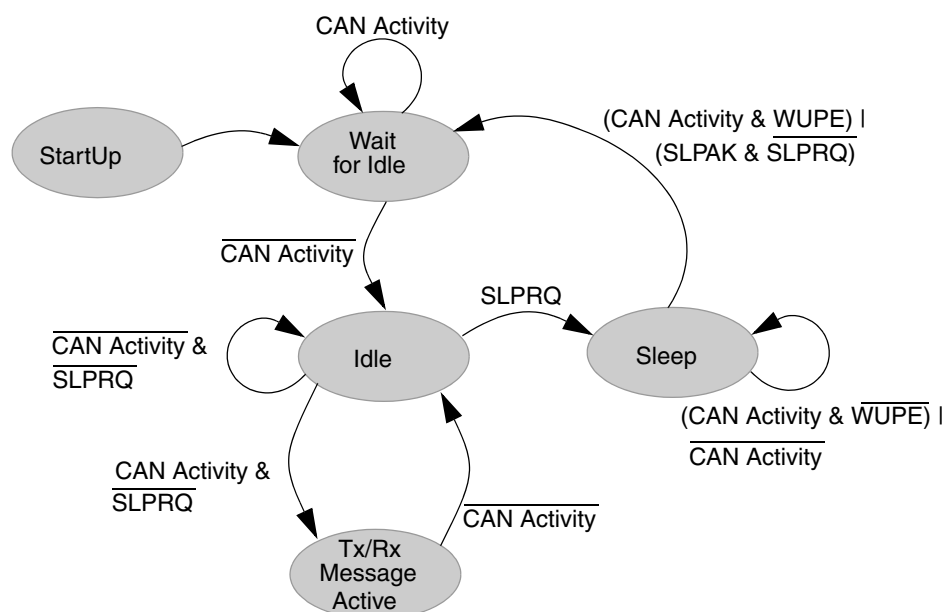


Figure 25-42. Simplified State Transitions for Entering/Leaving Sleep Mode

25.4.8.2 MSCAN Initialization Mode

In initialization mode, any ongoing transmission or reception is immediately aborted and synchronization to the bus is lost potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE

Ensure the MSCAN is not active when initialization mode is entered. The recommended procedure is to bring the MSCAN into sleep mode (SLPRQ=1 and SLPAK=1) before setting the INITRQ bit in the CANCTL0 register. Otherwise, the abort of an ongoing message can cause an error condition and can have an impact on the other bus devices.

In initialization mode, the MSCAN is stopped. However, interface registers can continue to be accessed. This mode resets the CANTCTL0, CANRFLG, CANIER, CANTFLG, CANTIER, CANTARQ,

CANTAACK, and CANTBSEL registers to their default values. In addition, it enables the configuration of the CANBTR0 and CANBTR1 bit timing registers, CANIDAC, and the CANIDAR and CANIDMR message filters. [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\),”](#) for a detailed description of the initialization mode.

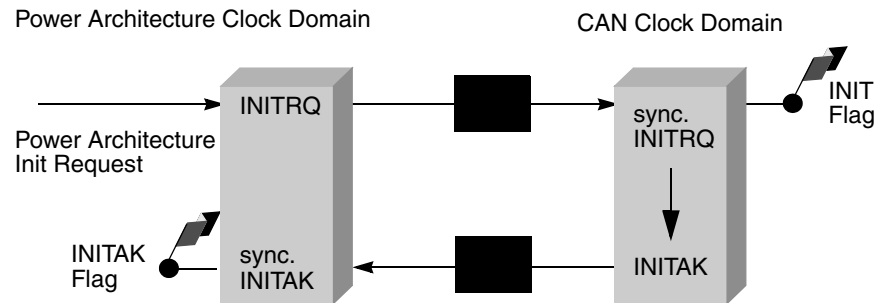


Figure 25-43. Initialization Request/Acknowledge Cycle

Due to independent clock domains within the MSCAN, INITRQ must be synchronized to all domains by using a special handshake mechanism. This handshake causes additional synchronization delay (see [Section Figure 25-43., “Initialization Request/Acknowledge Cycle”](#)).

If there is no message transfer ongoing on the CAN bus, the minimum delay is two additional bus clocks and three additional CAN clocks. When all parts of the MSCAN are in initialization mode, the INITAK flag is set. The application software must use INITAK as a handshake indication for the request (INITRQ) to go into initialization mode.

NOTE

The Power Architecture cannot clear the INITRQ bit before initialization mode (INITRQ=1 and INITAK=1) is active.

25.4.8.3 MSCAN Power Down Mode

The MSCAN is in power down mode ([Table 25-33](#)) when

- Power Architecture is in deep sleep mode

When entering the power down mode, the MSCAN immediately stops all ongoing transmissions and receptions, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations to the above rule, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE

You are responsible for ensuring that the MSCAN is not active when Power Architecture deep sleep mode is entered. The recommended procedure is to bring the MSCAN into Sleep mode before the Power Architecture enters deep sleep mode. Otherwise, the abort of an ongoing message can cause an error condition and impact other CAN bus devices.

In power down mode, all clocks are stopped and no registers can be accessed. If the MSCAN was not in sleep mode before power down mode became active, the module would perform an internal recovery cycle after powering up. This causes some fixed delay before the module enters run mode again.

25.4.8.4 Programmable Wake-Up Function

The MSCAN can be programmed to wake-up the MSCAN as soon as bus activity is detected (see control bit WUPE in [Section 25.3.2.1, “MSCAN Control 0 Register \(CANCTL0\)”](#)).

25.4.9 Reset Initialization

The reset state of each individual bit is listed in [Section 25.3.2, “Register Descriptions,”](#) which details all the registers and their bit-fields.

25.4.10 Interrupts

This section describes all interrupts originated by the MSCAN. It documents the enable bits and generated flags. Each interrupt is listed and described separately.

25.4.11 Description of Interrupt Operation

The MSCAN supports one interrupt vector mapped onto eight different interrupt sources, any of which can be individually masked (for details see sections [Section 25.3.2.6, “MSCAN Receiver Interrupt Enable Register \(CANRIER\)”](#) to [Section 25.3.2.8, “MSCAN Transmitter Interrupt Enable Register \(CANTIER\)”](#)).

NOTE

The dedicated interrupt vector addresses are defined in the [Resets and Interrupts](#) chapter.

Table 25-34. Interrupt Vectors

Interrupt Source	CCR Mask	Local Enable
Wake-Up Interrupt (WUPIF)	1 bit	CANRIER (WUPIE)
Error Interrupts Interrupt (CSCIF, OVRIF)	1 bit	CANRIER (CSCIE, OVRIE)
Receive Interrupt (RXF)	1 bit	CANRIER (RXFIE)
Transmit Interrupts (TXE[2:0])	1 bit	CANTIER (TXEIE[2:0])

25.4.11.1 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. The TXEx flag of the empty message buffer is set.

25.4.11.2 Receive Interrupt

A message is successfully received and shifted into the foreground buffer (RxFG) of the receiver FIFO. This interrupt is generated immediately after receiving the EOF symbol. The RXF flag is set. If there are multiple messages in the receiver FIFO, the RXF flag is set as soon as the next message is shifted to the foreground buffer.

25.4.11.3 Wake-Up Interrupt

Activity on the CAN bus occurred during MSCAN internal sleep mode and WUPE [Section 25.3.2.1](#), “MSCAN Control 0 Register (CANCTL0)” enabled.

25.4.11.4 Error Interrupt

An error interrupt is generated if an overrun of the receiver FIFO, error, warning, or bus-off condition occurs. [Section 25.3.2.5](#), “MSCAN Receiver Flag Register (CANRFLG)” indicates one of the following conditions:

- Overrun – An overrun condition of the receiver FIFO as described in [Section 25.4.2.3](#), “Receive Structures,” occurred.
- CAN Status Change – The actual value of the transmit and receive error counters control the CAN bus state of the MSCAN. As soon as the error counters skip into a critical range (Tx/Rx-warning, Tx/Rx-error, bus-off) the MSCAN flags an error condition. The status change, which caused the error condition, is indicated by the TSTAT and RSTAT flags (see [Section 25.3.2.5](#), “MSCAN Receiver Flag Register (CANRFLG)” and [Section 25.3.2.6](#), “MSCAN Receiver Interrupt Enable Register (CANRIER)”).

25.4.12 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in either the [Section 25.3.2.5](#), “MSCAN Receiver Flag Register (CANRFLG)” or the [Section 25.3.2.7](#), “MSCAN Transmitter Flag Register (CANTFLG)”. Interrupts are pending as long as one of the corresponding flags is set. The flags in the above registers must be reset within the interrupt manager to handshake the interrupt. The flags are reset by writing a 1 to the corresponding bit position.

NOTE

It must be guaranteed that the Power Architecture only clears the bit causing the current interrupt. RXFIF interrupt clear has the side effect of moving the receive buffer from the current to the next. Clearing a TXExIF flag also clears the corresponding ABTAKx. When a TXExIF flag is set, the corresponding ABTRQx bit is cleared

25.4.13 Recovery from Deep Sleep Mode

The MSCAN can wake up the Power Architecture from deep sleep mode via the wake-up interrupt. This interrupt can only occur if the MSCAN is in Sleep Mode (SLPRQ=1 and SLPK=1), the wake-up option is enabled (WUPE=1), and the wake-up interrupt is enabled (WUPIE=1).

When there is recessive to dominant change in CAN_RX pin, there is a wake-up interrupt generated asynchronously. Power Architecture can then be waken up from deep sleep mode by that interrupt.

After Power Architecture be waken up from deep sleep mode, it can enable the clocks of MSCAN and put MSCAN into normal operation mode again.

NOTE

Only MSCAN modules 1 and 2 can wake up the MPC5121 from deep sleep. MSCAN modules 3 and 4 cannot wake up the MPC5121 from deep sleep mode.

25.4.14 MSCAN Initialization

The procedure to initially start up the MSCAN module out of reset is as follows:

1. Assert CANE
2. Write to the configuration registers in initialization mode
3. Clear INITRQ to leave initialization mode and enter normal mode

If the configuration of registers which are writable in initialization mode only needs to be changed when the MSCAN module is in normal mode:

1. Make sure that the MSCAN transmission queue becomes empty and bring the module into sleep mode by asserting SLPRQ and awaiting SLPK
2. Enter initialization mode: Assert INITRQ and await INITAK
3. Write to the configuration registers in initialization mode
4. Clear INITRQ to leave initialization mode and continue in normal mode

25.4.15 Bus-Off Recovery

The bus-off recovery is configurable: the bus-off state can be left automatically or on request.

For reasons of backwards compatibility the MSCAN defaults to automatic recovery after reset. In this case, the MSCAN becomes error active again after counting 128 occurrences of 11 consecutive recessive bits on the CAN bus.

These two events may occur in any order.

Chapter 26

NAND Flash Controller (NFC)

26.1 Introduction

Composed of various control logic units and a 4 kilobyte internal RAM buffer, the NAND flash Controller implements the interface to standard NAND flash memory devices. Figure 26-1 shows the block diagram of NFC.

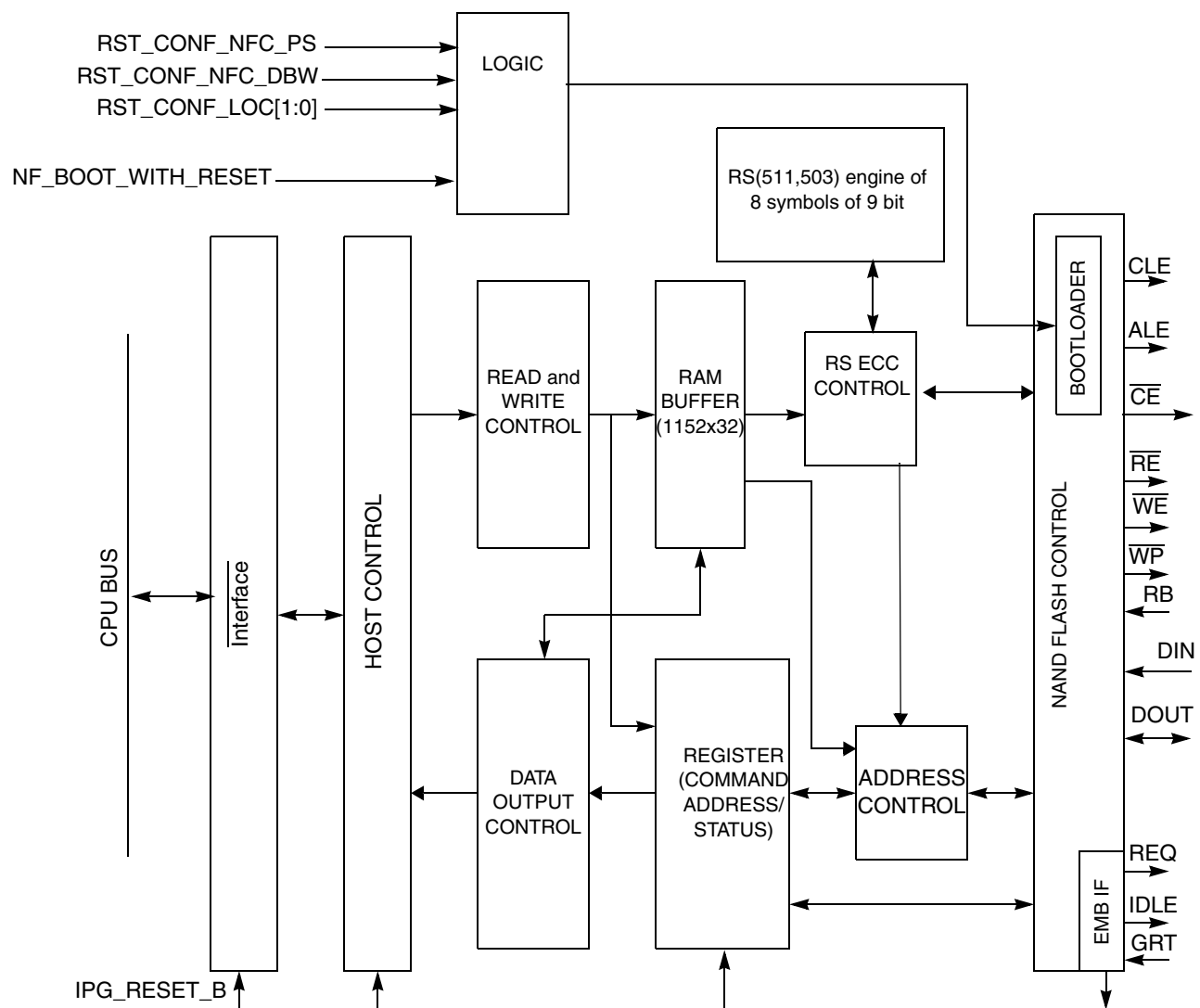


Figure 26-1. NAND Flash Controller Block Diagram

26.2 Overview

The NAND flash controller implements the interface to standard NAND flash memory devices. It is comprised of control logic and 4 KB internal RAM buffer. This 4 KB RAM buffer can be used as Boot RAM during cold reset (If boot from NAND flash is chosen), and is used as Buffer RAM after boot procedure. The controller supports NAND flash devices with page sizes of 512 B/2 KB/4 KB.

Normal operation for reading NAND flash consists of the following sequence:

1. Configure NFC
2. Request data block
3. Wait for interrupt
4. copy data from NFC to RAM

Operation for the write sequence follows a similar sequence. This sequence is as follows:

1. Assume NFC is configured.
2. Write data block into buffer
3. initiate program command
4. Wait for interrupt
5. Verify write occurred with no errors.

26.3 Features

The NAND flash controller includes the following features:

- Organization
 - NAND flash interface: 8-bits/16-bits (Pin Option)
- Internal RAM buffer(4K Bytes + 512Bytes)
 - Boot RAM at booting, buffer at normal operation
 - Memory mapped registers and internal RAM buffer
- Support large and small NAND flash.
 - Supports all NAND flash products of up to 64 K blocks (i.e, for SLC, 1/2 K page – 16 KB perblock, NFC supports up to 8 GBit Devices, for SLC 2 K page – 128 KB per block, NFC supports up to 64 Gbit devices, and for MLC 4K page – 512KB per block, NFC supports up to 256 Gbit devices.
- MLC NAND flash support by using 2 Reed Solomon RS(511,503) error correction codes (corrects 4/8 error bits in 528/538 bytes (512Bytes main+16/26Bytes spare). Correction capability is configurable.
- DMA request
 - DMA request for a page/section read and other read operations. After DMA read occurs the request is de-asserted after the first read of the NFC page buffer.
- ECC mode/Bypass ECC
- Internal Boot code loader during power-up (that can be enabled/disabled)
 - Data Protection

- Write Protection mode for RAM buffer: Write protection of RAM buffer (LSB 2KB of RAM buffer)
- Write Protection mode for NAND flash: block based write protection of NAND flash
- Write protection for RAM buffer and NAND flash during power-up
- Handshaking Feature
 - INT pin: Indicates end of flash operation.
- IO pins sharing support
 - Allow sharing of the IO pins with other memory controllers through special arbitration logic.

26.4 External Signal Description

Table 26-1. Signal Properties

Name	Function	I/O	Reset
NFC_DATA[15:0]	NFC data bus	I	xxxx
IPP_DO_NFC_CE0	Flash Chip Enable 0	O	1
IPP_DO_NFC_CE1	Flash Chip Enable 1	O	1
IPP_DO_NFC_CE2	Flash Chip Enable 2	O	1
IPP_DO_NFC_CE3	Flash Chip Enable 3	O	1
IPP_DO_NFC_RE	Flash Read Enable	O	1
IPP_DO_NFC_WE	Flash Write Enable	O	1
IPP_DO_NFC_CLE	Flash Command Latch Enable	O	0
IPP_DO_NFC_ALE	Flash Address Latch Enable	O	0
IPP_DO_NFC_WP	Flash Write Protect	O	1
IPP_IND_NFC_RB	Flash Ready/Busy	I	1

26.5 Memory Map and Register Definition

The register map is shown in [Table 26-2](#).

Table 26-2. NFC Register Map

Offset or Address	Register	Access	Reset Value	Section/Page
Internal RAM				
0x0000-0x11FF	Internal RAM	RW	—	—
General Registers				
0x1E00	Reserved	—	—	—
0x1E02	Reserved	—	—	—
0x1E04	RAM_BUF_ADDR	RW	0x0000	26.5.4.1/26-8
0x1E06	FLASH_ADDR	RW	0x0000	26.5.4.2/26-10
0x1E08	FLASH_CMD	RW	0x0000	26.5.4.3/26-10

Table 26-2. NFC Register Map (continued)

0x1E0A	NFC_CFG	RW	0x0001	26.5.4.4/26-11
0x1E0C	ECC_STATUS1	R	0x0000	26.5.4.5/26-12
0x1E0E	ECC_STATUS2	R	0x0000	26.5.4.6/26-13
0x1E10	SPAS	RW	0x006D	26.5.4.7/26-13
0x1E12	NF_WR_PROT	RW	0x0002	26.5.4.8/26-14
0x1E14	Reserved	—	0x0000	—
0x1E16	Reserved	—	0x0000	—
0x1E18	Flash_WP_ST	R	0x0492	26.5.4.9/26-14
0x1E1A	NF_CFG1	RW	0x0C2A	26.5.4.10/26-15
0x1E1C	NF_CFG2	RW	0x0000	26.5.4.11/26-17
0x1E20	NFC_USBA0	RW	0x0000	26.5.4.12/26-18
0x1E22	NFC_UEBA0	RW	0x0000	26.5.4.13/26-19
0x1E24	NFC_USBA1	RW	0x0000	26.5.4.14/26-19
0x1E26	NFC_UEBA1	RW	0x0000	26.5.4.15/26-20
0x1E28	NFC_USBA2	RW	0x0000	26.5.4.16/26-20
0x1E2A	NFC_UEBA2	RW	0x0000	26.5.4.17/26-21
0x1E2C	NFC_USBA3	RW	0x0000	26.5.4.18/26-21
0x1E2E	NFC_UEBA3	RW	0x0000	26.5.4.19/26-22

26.5.1 Internal RAM

[Table 26-3](#) shows the organization of the buffer memory space in the NFC.

Table 26-3. Data (Buffer) Organization in Memory

Address	Use	Access
0000 – 01FF	Main area Buffer 0	R/W
0200 – 03FF	Main area Buffer 1	R/W
0400 – 05FF	Main area Buffer 2	R/W
0600 – 07FF	Main area Buffer 3	R/W
0800 – 09FF	Main area Buffer 4	R/W
0A00 – 0BFF	Main area Buffer 5	R/W
0C00 – 0DFF	Main area Buffer 6	R/W
0E00 – 0FFF	Main area Buffer 7	R/W
1000 – 103F	Spare area Buffer 0	R/W
1040 – 107F	Spare area Buffer 1	R/W
1080 – 10BF	Spare area Buffer 2	R/W
10C0 – 10FF	Spare area Buffer 3	R/W

Table 26-3. Data (Buffer) Organization in Memory (continued)

Address	Use	Access
1100 – 113F	Spare area Buffer 4	R/W
1140 – 117F	Spare area Buffer 5	R/W
1180 – 11BF	Spare area Buffer 6	R/W
11C0 – 11FF	Spare area Buffer 7	R/W

26.5.2 Spare Area Buffer

The main area buffer is a general data block. The spare area buffer is used for a variety of functions including Error Correction. [Table 26-4](#) shows the spare area mapping inside the internal buffer. The host can use all of the spare area except for the ECC code area.

NFC automatically generates ECC code for both main and spare area during data programming to NAND flash, but does not update ECC code to spare buffer. When programming/reading spare area, the spare area buffer number (SB0–SB7) must be selected using the RAM buffer address register (RAM_BUF_ADDR).

Table 26-4. Spare Area Buffer

Address	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
1000h (SB0)	LSN(2nd) ¹								LSN(1st)							
1002h (SB0)	WC(1st) ²								LSN(3rd)							
1004h (SB0)	BI ³								WC(2nd)							
1006h (SB0)	RS ECC Code for Main & Spare area data (1st)								Reserved							
1008h (SB0)	ECC Code for Main area data (3rd)								ECC Code for Main area data (2nd)							
100Ah (SB0)	ECC Code for Main area data (5th)								ECC Code for Main area data (4th)							
100Ch (SB0)	ECC Code for Main area data (7th)								ECC Code for Main area data (6th)							
100Eh (SB0)	ECC Code for Main area data (9th)								ECC Code for Main area data (8th)							
1010h (SB0)	ECC Code for Main area data (11rd) ⁴								ECC Code for Main area data (10nd) ⁴							
1012h (SB0)	ECC Code for Main area data (13th) ⁴								ECC Code for Main area data (12th) ⁴							

Table 26-4. Spare Area Buffer

1014h (SB0)	ECC Code for Main area data (15th) ⁴	ECC Code for Main area data (14th) ⁴
1016h (SB0)	ECC Code for Main area data (17th) ⁴	ECC Code for Main area data (16th) ⁴
1018h (SB0)	Reserved	ECC Code for Main area data (16th) ⁴
101Ah– 103Eh (SB0)	Reserved	
1040h– 107Eh (SB1)	SB1–SB7 have same assignment like SB0.	
1080h– 10BEh (SB2)		
10C0h– 10FEh (SB3)		
1100H– 113EH (SB4)		
1140H– 117EH (SB5)		
1180H– 11BEH (SB6)		
11C0H– 11FEH (SB7)		

¹ LSN: Logical Sector Number² WC: Wrap Count and other bytes has same wrap count information and those are used as error correction for wrap count itself.³ BI: Bad Block Information⁴ ECC: These ECC Bytes can be used only if the NAND device has enough spare area to store all ECC data. Generally, such mode is supported only by NAND flash with 4096 bytes main area plus 218 bytes spare area in page size.

26.5.3 Register Summary

Table 26-5. NFC Register Summary

Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RAM_BUF_ADDR (0xBASE + 0x1E04)	R	0	0	0	0	0	0	0	0	0	0	ACTIVE_CS		0	RBA		
	W																
FLASH_ADDR (0xBASE + 0x1E06)	R	ADD															
	W																
FLASH_CMD (0xBASE + 0x1E08)	R	CMD															
	W																
NFC_CFG (0xBASE + 0x1E0A)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BLS	
	W															BLS	
ECC_STATUS1 (0xBASE + 0x1E0C)	R	NOSER4				NOSER3				NOSER2				NOSER1			
	W																
ECC_STATUS2 (0xBASE + 0x1E0E)	R	NOSER8				NOSER7				NOSER6				NOSER5			
	W																
SPAS (0xBASE + 0x1E10)	R	0	0	0	0	0	0	0	0	SPAS							
	W																
NF_WR_PROT (0xBASE + 0x1E12)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	WPC		
	W																
Flash_WP_ST (0xBASE + 0x1E18)	R	0	0	0	0	US3	LS3	LTS3	US2	LS2	LTS2	US1	LS1	LTS1	US0	LS0	LTS0
	W																

Table 26-5. NFC Register Summary (continued)

Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NF_CFG1 (0xBASE + 0x1E1A)	R	0	0	0	IGN _RB	FP_I NT	PPB		SYM	NF_ CE	RST	NF_BI G	INT_ MSK	ECC _EN	SP_ EN	dma_ mode	ecc_m ode
	W																
NF_CFG2 (0xBASE + 0x1E1C)	R	INT	CM D_F AIL	0	0	0	0	0	0	0	0	FDO			FDI	FAD D	FCMD
	W																
NFC_USBA0 (0xBASE + 0x1E20)	R	USBA0															
	W																
NFC_UEBA0 (0xBASE + 0x1E22)	R	UEBA0															
	W																
NFC_USBA1 (0xBASE + 0x1E24)	R	USBA1															
	W																
NFC_USBA1 (0xBASE + 0x1E26)	R	UEBA1															
	W																
NFC_USBA2 (0xBASE + 0x1E28)	R	USBA2															
	W																
NFC_UEBA2 (0xBASE + 0x1E2A)	R	UEBA2															
	W																
NFC_USBA3 (0xBASE + 0x1E2C)	R	USBA3															
	W																
NFC_UEBA3 (0xBASE + 0x1E2E)	R	UEBA3															
	W																

26.5.4 Register Descriptions

26.5.4.1 Buffer Number for Page Data Transfer(RAM_BUF_ADDR)

RAM_BUF_ADDR specifies which part of the RAM Buffer is transferred to/from flash memory and with NAND flash device is currently used. The bit assignments for the register are shown in [Figure 26-2](#) and the field descriptions are shown in [Table 26-6](#).

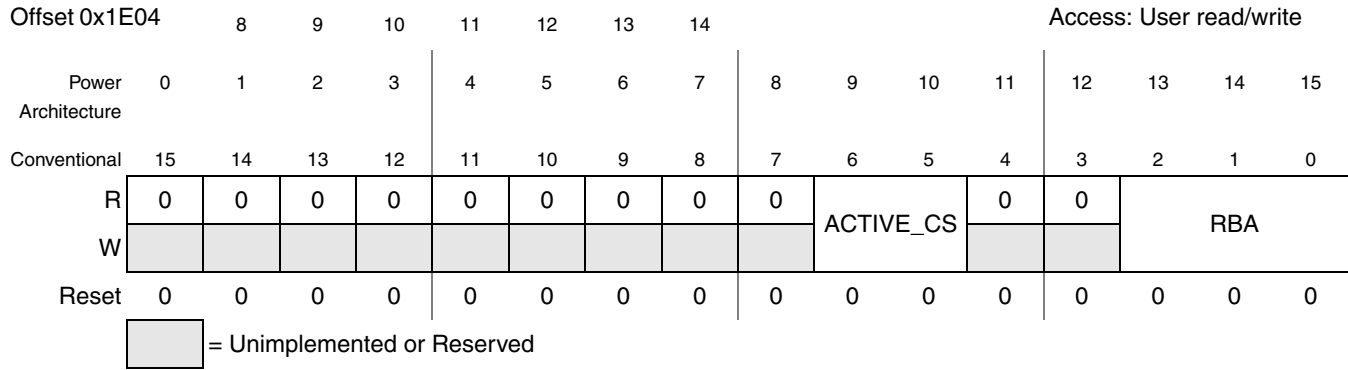


Figure 26-2. RAM_BUF_ADDR Register

Table 26-6. RAM_BUF_ADDR Register Field Descriptions

Field	Description
ACTIVE_CS	Active Chip Select. Defines the chip-select line to be asserted during any NAND operation. 00 chip enable 0 is selected 01 chip enable 1 is selected 10 chip enable 2 is selected 11 chip enable 3 is selected
RBA	Specifies what part of RAM Buffer is transferred to/from flash memory. 000 1st internal RAMbuffer is selected 001 2nd internal RAMbuffer is selected 010 3rd internal RAMbuffer is selected 011 4th internal RAMbuffer is selected 100 5th internal RAMbuffer is selected 101 6nd internal RAMbuffer is selected 110 7rd internal RAMbuffer is selected 111 8th internal RAMbuffer. is selected

26.5.4.2 NAND Flash Address Register (FLASH_ADDR)

The NAND flash Address (FLASH_ADDR) register is a read-write register containing the address of the NAND flash device that is read, programmed or erased. The address in the FLASH_ADDR register is written to the flash device. The bit assignments for the register are shown in [Figure 26-3](#) and the field descriptions are shown in [Table 26-7](#).

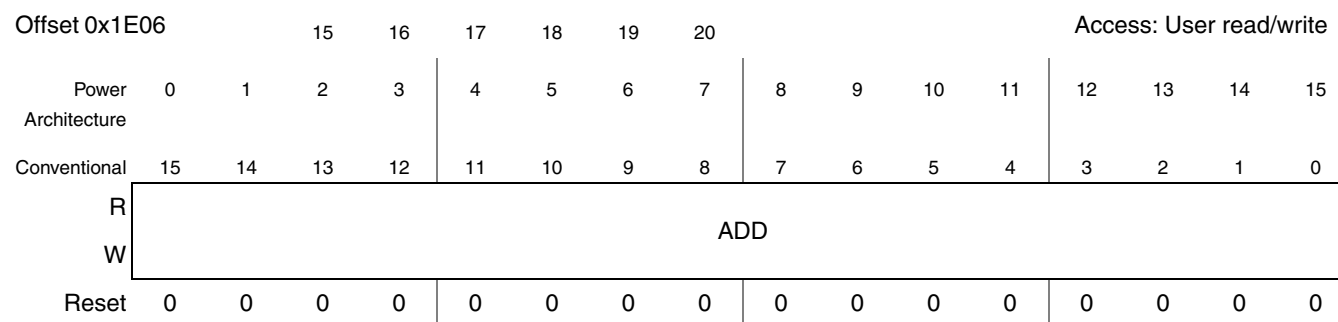


Figure 26-3. FLASH_ADDR Register

Table 26-7. FLASH_ADDR Register Field Descriptions

Field	Description
ADD	NAND Flash Address. NAND flash address that are read programmed or erased. This ADD is entered into NAND flash device.

26.5.4.3 NAND Flash Command Register (FLASH_CMD)

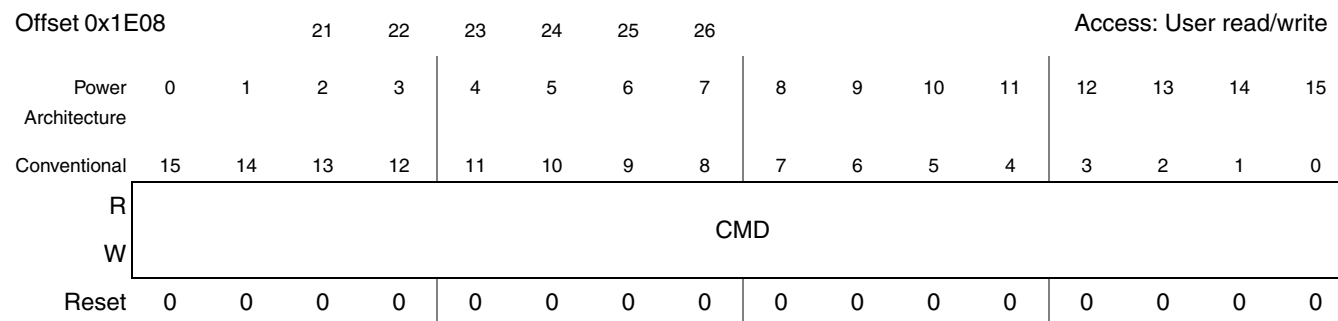


Figure 26-4. FLASH_CMD Register

Table 26-8. FLASH_CMD Register Field Descriptions

Field	Description
CMD	NAND flash command. This CMD is the command that is entered into NAND Flash.

26.5.4.4 NAND Flash Controller Internal Buffer Lock Control(NFC_CFG)

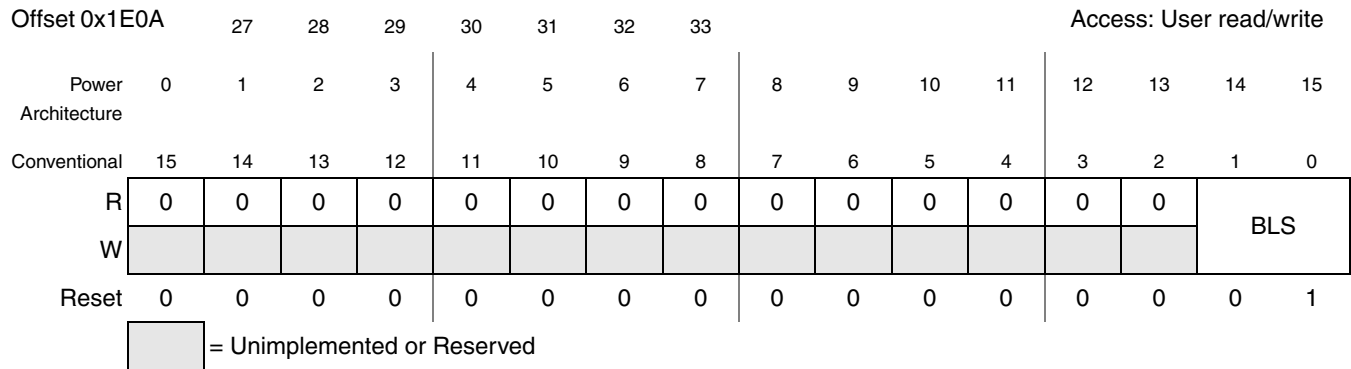


Figure 26-5. NFC_CFG Register

Table 26-9. NFC_CFG Register Field Descriptions

Field	Description
BLS	Buffer Lock Set. This field specifies the buffer lock status of first 4 sections in the internal buffer. The other 4 sections are always Unlocked. 00 Locked 01 Locked (default) 10 Unlocked 11 Locked

26.5.4.5 Controller Status and the Result of Flash Operation (ECC_STATUS1)

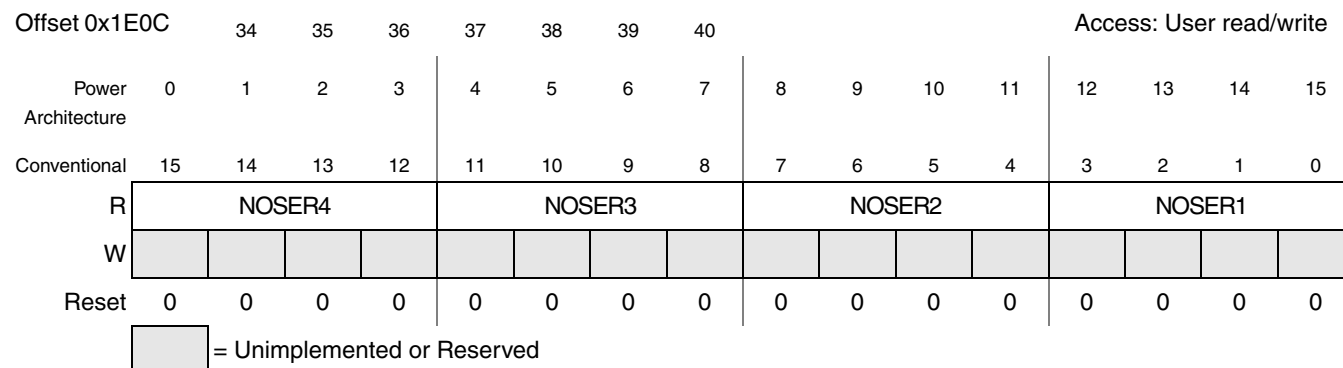


Figure 26-6. RECC_STATUS1 Register

Table 26-10. RECC_STATUS1 Register Field Descriptions

Field	Description
NOSER4	Number Of symbol Errors for fourth section of the internal RAM (528/538 bytes based on ecc_mode). This field shows the number of symbols with errors in 512 bytes main plus 16/26 bytes spare (totally 528/538 bytes) as a result of the RS(511,503) ECC check upon read. For the entire 528/538 bytes the number of correctable bits is determined by ecc_mode. 0000 No error 0001 – 1000 >Indicates the number of symbol Errors (Correctable Error) 1111 Error count exceed the ECC capability Others Reserved
NOSER3	Same as NOSER4, this is number of symbol error for third section.
NOSER2	Same as NOSER4, this is number of symbol error for second section.
NOSER1	Same as NOSER4, this is number of symbol error for first section.

26.5.4.6 Controller Status and the Result of Flash Operation (ECC_STATUS2)

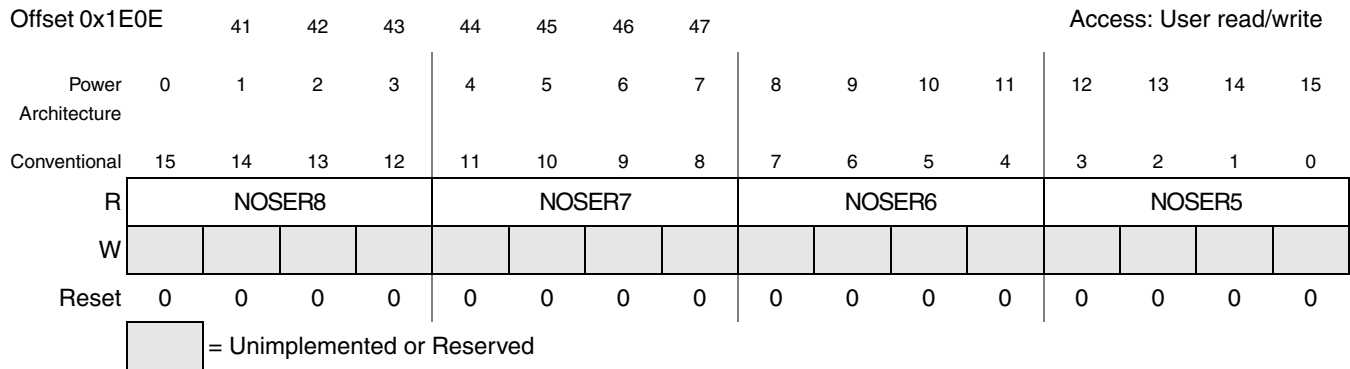


Figure 26-7. ECC_STATUS2 Register

Table 26-11. ECC_STATUS2 Register Field Descriptions

Field	Description
NOSER8	Number Of symbol Errors for eighth section of the internal RAM (528/538 bytes based on ecc_mode). This field shows the number of symbols with errors in 512 bytes main plus 16/26 bytes spare (totally 528/538 bytes) as a result of the RS(511,503) ECC check upon read. For the entire 528/538 bytes the number of correctable bits is determined by ecc_mode. 0000 No error 0001 – 1000 >Indicates the number of symbol Errors (Correctable Error) 1111 Error count exceed the ECC capability. Others Reserved
NOSER7	Same as NOSER8, this is number of symbol error for seventh section.
NOSER6	Same as NOSER8, this is number of symbol error for sixth section.
NOSER5	Same as NOSER8, this is number of symbol error for fifth section.

26.5.4.7 SPare Area Size (SPAS)

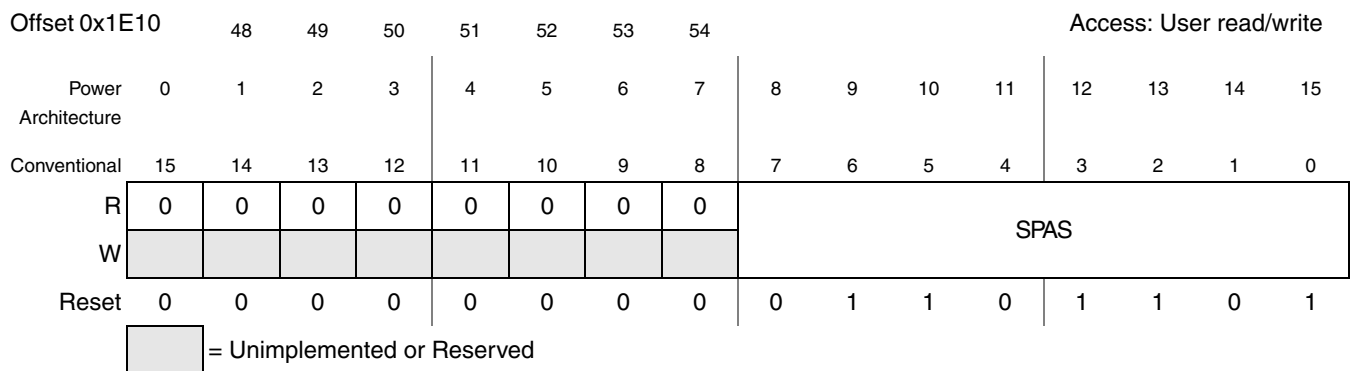


Figure 26-8. SPAS Register

Table 26-12. SPAS Register Field Descriptions

Field	Description
SPAS	Spare Area Size. This field specifies the size of the spare area of the NAND device. The size is in half-words. The size refers to a full page. i.e for 2K SLC device, SPAS should be set to 32. In boot mode, this field is overridden to correct one. but you do not see this override.

26.5.4.8 Nand Flash Write Protection (NF_WR_PROT)

Offset 0x1E12																Access: User read/write			
	55	56	57	58	59	60	61												
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	0	0	0	0	0	0	0	0	0	0	0	0	0	WPC					
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0			
	= Unimplemented or Reserved																		

Figure 26-9. NF_WR_PROT Register

Table 26-13. NF_WR_PROT Field Descriptions

Field	Description
WPC	Write Protection Command. The Command field specifies the operation which the controller performs. The command is performed on the protection mechanism of the chip-enable that is configured in active_cs field. If blocks are lock or lock-tight, command 0x10, 0x15 and 0xD0 is ignored, an command fail interrupt is generated to notify user. 100: Unlock NAND Flash block(s) according to NFC_USBA _n and NFC_UEBA _n , where <i>n</i> is defined as active_cs. 010: Lock all NAND Flash block(s) 001: Lock-tight locked block(s)

26.5.4.9 NAND Flash Write Protection Status (Flash_WP_ST)

Offset 0x1E18																Access: User read/write			
	62	63	64	65	66	67	68												
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	0	0	0	0	US3	LS3	LTS3	US2	LS2	LTS2	US1	LS1	LTS1	US0	LS0	LTS0			
W																			
Reset	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0			
	= Unimplemented or Reserved																		

Figure 26-10. Flash_WP_ST Register

Table 26-14. Flash_WP_ST Field Descriptions

Field	Description
US n	Unlocked Status. Specifies whether there are any unlocked blocks in the NAND Flash. 0 No unlocked block in NAND Flash 1 There are unlocked block(s) in NAND Flash where n is the the Device controlled by CE n .
LS n	Locked Status. Specifies that all NAND Flash blocks are in locked status. 0 Not all NAND Flash blocks are in locked status 1 All NAND Flash blocks are in locked status where n is the the Device controlled by CE n .
LTS n	Lock-tighten Status. Specifies that Locked block(s) is (are) lock-tightened. 0 Locked block(s) is not lock-tightened 1 Locked block(s) is lock-tightened where n is the the Device controlled by CE n .

Table 26-15. Write Protect Modes

State	Status Bits – US/LS/LTS
Lock – all blocks are locked	010
Unlock-lock – there are unlocked blocks	110
Unlock-Lockt – there are unlocked blocks: cant change to other state	101
Lockt – all block are locked: cant change to other state	001

26.5.4.10 NAND Flash Operation Configuration (NF_CFG1)

Offset 0xE1A								Access: User read/write								
	69	70	71	72	73	74	75									
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	IGN_RB	FP_INT	PPB		SYM	NF_CE	RST	NF_BIG	INT_MSK	ECC_EN	SP_EN	DMA_MODE	ECC_MODE
W																
Reset	0	0	0	0	1	1	0	0	0	0	1	0	1	0	1	0
	= Unimplemented or Reserved															

Figure 26-11. NF_CFG1 Register

Table 26-16. NF_CFG1 Field Descriptions

Field	Description
IGN_RB	Ignore handshake Ready/Busy from NAND flash. command send out and generate an interrupt immediatly, dont care if device is ready or not. 0 command send out and then check ready/busy, generate interrupt until NAND flash is in ready state. 1 command send out and generate interrupt, ignore the ready/busy status of NAND flash.
FP_INT	Full Page interrupt. NFC generated interrupt (during progra./read) after each section of 512B (plus accompanied spare bytes).or full page. If this bit is set, the interrupt is generated only after the whole page was read/programmed. This bit affects the interrupt only during read or program of a page. 0 NFC generates interrupt after each section of 512B plus accompanied spare bytes. 1 NFC generates interrupt only after the whole operation was completed.
PPB	Pages Per Block. Indicates how many pages are in 1 Block of the NAND flash. 00 32 pages per block 01 64 pages per block 10 128 pages per block 11 256 pages per block
SYM	Symmetric mode. 1 enable one internal clock (FLASH_CLK) cycle per access of RE# and WE#, (symmetric mode). 0 enable two internal clock (FLASH_CLK) cycles per access of RE# and WE#, (asymmetric mode). refer 26.7.2/26-48 for more information.
NF_CE	NAND Flash Force CE. This bit forces the CE# signal to the NAND Flash device to 0 when enabled. This bit allows a greater range of support new NAND Flash devices. 0 CE# signal operates normally 1 CE# signal is asserted as long as this bit is set to 1.
RST	NFC Reset. This bit resets the NFC state machine. This bit is self-cleared, and resets NF_BIG, INT_MSK and SP_EN bit at the same time. 0 Do not reset the NFC state machine 1 Reset the NFC state machine Note: If this reset happens, the current operation has ceased. To send the next command to flash, the software must send the reset command (0xFF) to flash first because the external flash model did not know the operation stopped.
NF_BIG	NAND FLASH Big Endian Mode. This bit enables big Endian mode when writing from internal RAM to the NAND flash device or reading from NANDflash device to internal RAM. 0 Little Endian mode 1 big Endian mode
INT_MSK	Mask interrupt Bit. This bit enables the interrupt by masking or not masking the interrupt bit. 0 Mask interrupt is disabled (interrupt enabled) 1 Mask interrupt is enabled (interrupt disabled)
ECC_EN	ECC operation Enable. This field determines whether ECC operation is executed or bypassed. 0 ECC operation is bypassed 1 ECC operation is executed
SP_EN	NAND Flash Spare Enable. This bit determines whether host reads/writes are to NAND flash spare data only or NAND flash main and spare data. This fetures is for some old NAND flash with 1/2K page size. There is no ECC in this mode of operation. 0 NAND flash main and spare data is enabled 1 NAND flash spare only data is enabled

Table 26-16. NF_CFG1 Field Descriptions (continued)

Field	Description
DMA_MODE	This bit defines the dma_req signal mode of operation during a page read. dma_req can be asserted after one section of the page is read (main + relevant part of the spare), or only at the end of the page read. Other read operations that assert dma_req are not affected by this bit. 0 dma_req is asserted after each section is read out. 1 dma_req is asserted only at the end of a page read.
ECC_MODE	This bit selects the ECC capabilities. Error correction mechanism can fix 4bit errors or 8bit errors. In 4bit ecc mode, the NFC uses 16Bytes of spare area for every 512Bytes section of the NAND device (7Bytes for user-specific application and 9Bytes for the ECC). reference to Table 26-4 for more information. In 8bit ecc mode, the NFC uses 26Bytes of spare area for every 512Bytes section of the NAND device (7Bytes for user-specific application, 18Bytes for the ECC and 1 reserved Byte). reference to Table 26-4 for mor information. 0 8bit error correction. 1 4bit error correction. To use 8-bit ECC mode, user must have a NAND flash that has at least 26Bytes of spare area per 512B main section.

26.5.4.11 NAND Flash Operation Configuration (NF_CFG2)

Offset 0x1E1C																Access: User read/write			
	76	77	78	79	80	81	82												
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	INT	CMD_FAIL	0	0	0	0	0	0	0	0	FDO		FDI	FADD	FCMD				
W																			
Reset	0 ¹	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

1. INT (Bit 15) reset value is ZERO, but soon after powerup it changes to ONE. When performing boot from NAND FLASH, the INT bit changes from ZERO to ONE after the transfer of boot code has been accomplished. For more information refer to [Section 26.6.1, “Modes of Operation”](#).


 = Unimplemented or Reserved

Figure 26-12. NF_CFG2 Register

Table 26-17. NF_CFG2 Field Descriptions

Field	Description
INT	Interrupt. This field determines the state of the interrupt output of the controller. It is set by the controller when basic operation (FDI, FDO, FCMD or FADDR operation) is done. It is cleared by the Host by writing 0 to this field. (Host can also set this bit by writing 1 to this field). 0 Basic operation or bootloading continues running 1 Basic operation or boot loading is done
CMD_FAIL	Command fail interrupt. This field shows that command 0x10, 0x15 or 0xD0 is failed to be sent out. the reason of failure is try to program or erase locked blocks. check Section 26.5.4.8, “Nand Flash Write Protection (NF_WR_PROT)” , and unlock blocks. write 0 to clear this bit.

Table 26-17. NF_CFG2 Field Descriptions (continued)

Field	Description
FDO	NAND Flash Data Output. This field specifies the read data from NAND flash. 001 One page data out ¹ 010 NAND flash ID data out 100 NAND flash status register dataout other Not supported except all 0.
FDI	NAND Flash Data Input. This field specifies the program NAND flash. 0 No NAND flash data input operation 1 Activate NAND flash data input operation
FADD	Initial an address to NAND flash. 0 No NAND flash data input operation 1 Initial an address to NAND flash
FCMD ²	Initial a command to NAND flash. 0 No NAND flash data input operation 1 Initial a command to NAND flash

¹ Page size is determined by SP_EN register bit (main_spare or spare only).

² If try to access block which is lock/lock-tight, CMD register is 0x10, 0x15 or 0xD0, write 1 to FCMD bit is ignored and a command fail interrupt is generated.

NOTE

When the basic operation is completed, the FCMD/FADD/FDI/FDO bits change to LOW automatically.

Only one operation among FCMD/FADD/FDI/FDO can be activated.

26.5.4.12 Start Block Address to Unlock in Write Protection Mode (NFC_USBA0)

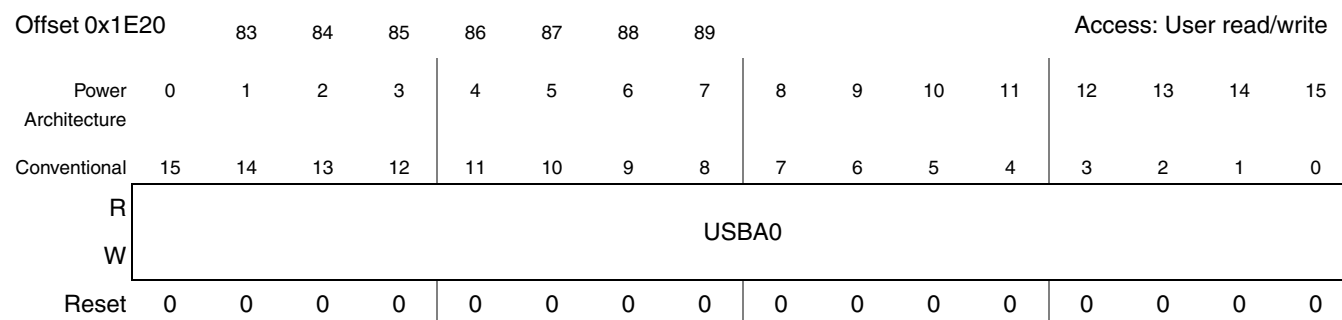


Figure 26-13. NFC_USBA0 Register

Table 26-18. NFC_USBA0 Field Descriptions

Field	Description
USBA0	Unlock Start Block Address for NAND flash controlled by CE0. Start NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.13 End Block Address to Unlock in Write Protection Mode (NFC_UEBA0)

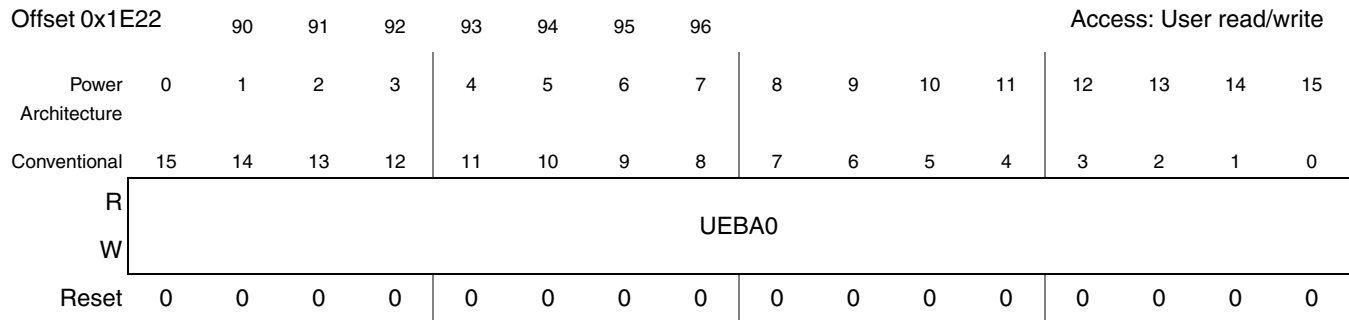


Figure 26-14. NFC_UEBA0 Register

Table 26-19. NFC_UEBA0 Field Descriptions

Field	Description
UEBA0	Unlock End Block Address of NAND flash controlled by CE0. End NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.14 Start Block Address to Unlock in Write Protection Mode (NFC_USBA1)

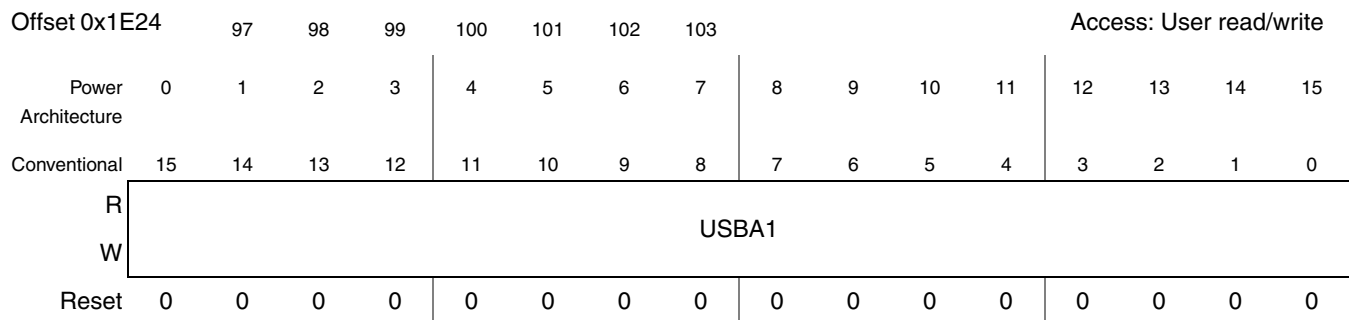


Figure 26-15. NFC_USBA1 Register

Table 26-20. NFC_USBA1 Field Descriptions

Field	Description
USBA1	Unlock Start Block Address for NAND Flash controlled by CE1. Start NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.15 End Block Address to Unlock in Write Protection Mode (NFC_UEBA1)

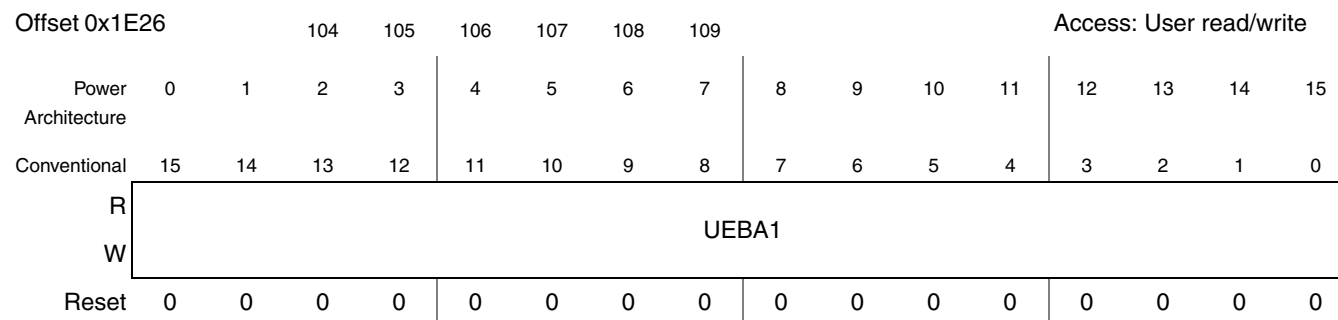


Figure 26-16. NFC_UEBA1 Register

Table 26-21. NFC_UEBA1 Field Descriptions

Field	Description
UEBA1	Unlock End Block Address of NAND flash controlled by CE0. End NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.16 Start Block Address to Unlock in Write Protection Mode (NFC_USBA2)

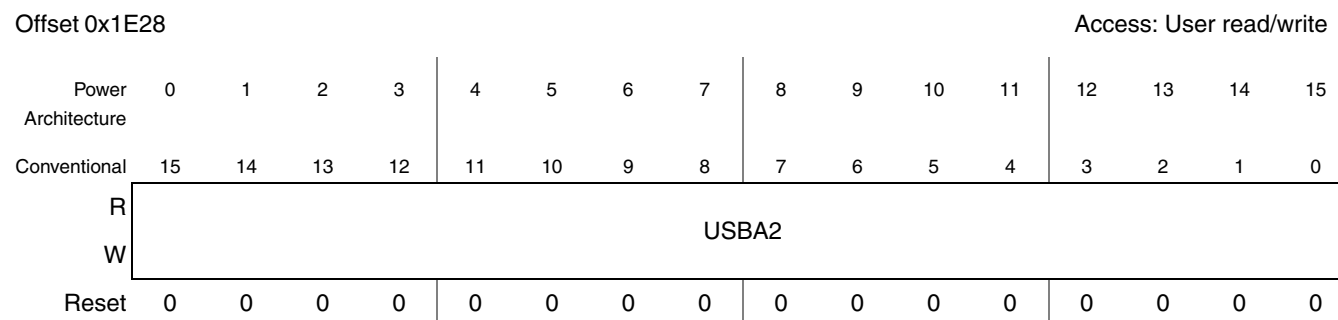


Figure 26-17. NFC_USBA2 Register

Table 26-22. NFC_USBA2 Field Descriptions

Field	Description
USBA2	Unlock Start Block Address for NAND flash controlled by CE2. Start NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.17 End Block Address to Unlock in Write Protection Mode (NFC_UEBA2)

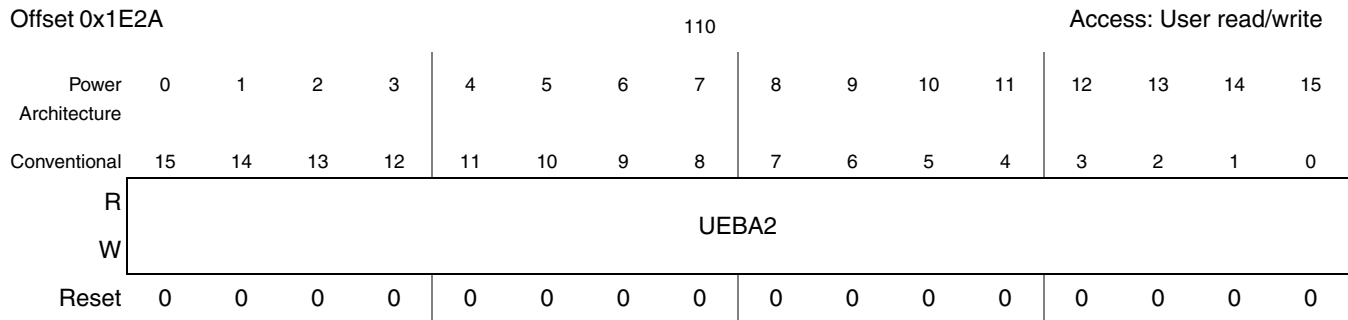


Figure 26-18. NFC_UEBA2 Register

Table 26-23. NFC_UEBA2 Field Descriptions

Field	Description
UEBA2	Unlock End Block Address of NAND flash controlled by CE3. End NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.18 Start Block Address to Unlock in Write Protection Mode (NFC_USBA3)

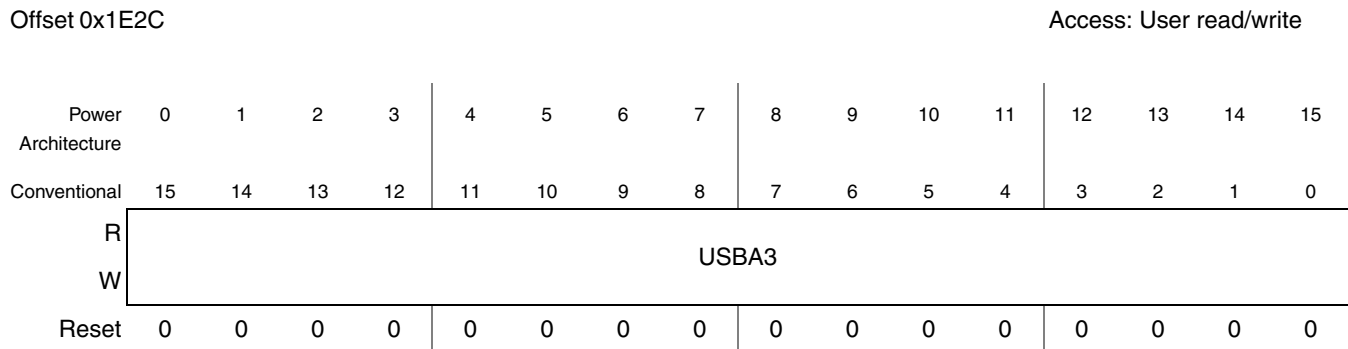


Figure 26-19. NFC_USBA3 Register

Table 26-24. NFC_USBA3 Field Descriptions

Field	Description
USBA3	Unlock Start Block Address for NAND flash controlled by CE3. Start NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.5.4.19 End Block Address to Unlock in Write Protection Mode (NFC_UEBA3)

Offset 0x1E2E

Access: User read/write

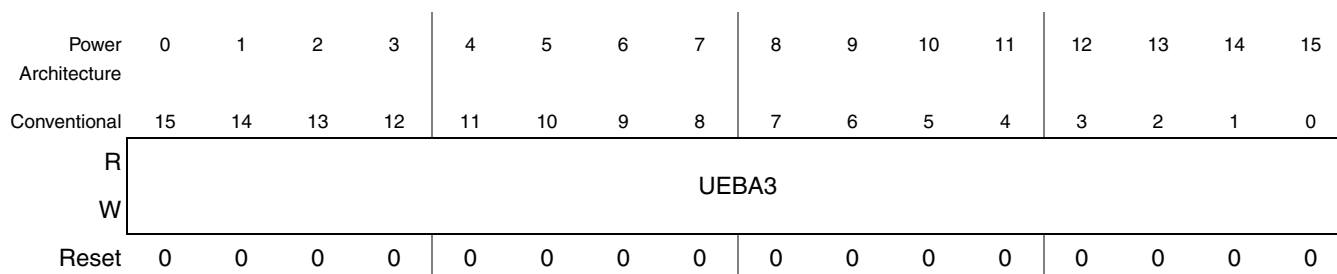


Figure 26-20. NFC_UEBA3 Register

Table 26-25. NFC_UEBA3 Field Descriptions

Field	Description
UEBA3	Unlock End Block Address of NAND flash controlled by CE3. End NAND flash block address to unlock in Write Protection mode, which follows Unlock block command.

26.6 Functional Description

26.6.1 Modes of Operation

The NAND FLASH Controller operating modes are described in this section. The operating mode is determined by four inputs: RST_CONF_NFC_PS, RST_CONF_NFC_DBW and RST_CONF_LOC[1:0], as shown in [Table 26-26](#).

The page size of NAND flash device is determined by RST_CONF_NFC_PS.

The bus width of NAND flash device is determined by the RST_CONF_NFC_DBW.

RST_CONF_LOC[1:0] is used to decide if boot ROM is in NFC.

Table 26-26. NAND FLASH Controller Operating Modes

RST_CONF_NFC_PS	RST_CONF_NFC_DBW	RST_CONF_LOC[1:0]	Function
1'b0	1'b0	2'b00	Do not Boot from NAND flash. NAND flash is configured to 8-bits I/O bus width and page size is 512 Bytes
1'b0	1'b1	2'b00	Do not Boot from NAND flash. NAND flash is configured to 16-bits I/O bus width and page size is 512 Bytes.
1'b1	1'b0	2'b00	Do not Boot from NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 2KB
1'b1	1'b1	2'b00	Do not Boot from NAND flash. NAND flash is configured to 16-bits I/O bus width and a page size is 2KB

Table 26-26. NAND FLASH Controller Operating Modes (continued)

RST_CONF_NFC_PS	RST_CONF_NFC_DBW	RST_CONF_L OC[1:0]	Function
1'bx	1'b0	2'b10	Do not Boot from NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 4KB
1'bx	1'b1	2'b10	Do not Boot from NAND flash. NAND flash is configured to 16-bits I/O bus width and a page size is 4KB
1'b0	1'b1	2'b11	Boot From x16 NAND flash. NAND flash is configured to 16-bits I/O bus width and a page size is 4KB + 128 bytes spare.Boot data is protected by 4-bit ECC mode.
1'b1	1'b1	2'b11	Boot From x16 NAND flash. NAND flash is configured to 16-bits I/O bus width and a page size is 4KB + 218 bytes spare.Boot data is protected by 8-bit ECC mode.
1'b0	1'b0	2'b11	Boot From x8 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 4KB + 128 bytes spare.Boot data is protected by 4-bit ECC mode.
1'b1	1'b0	2'b11	Boot From x8 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 4KB + 218 bytes spare.Boot data is protected by 8-bit ECC mode.
1'b0	1'b0	2'b01	Boot From x8 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 512 + 16 bytes spare.Boot data is protected by 4-bit ECC mode.
1'b0	1'b1	2'b01	Boot From x16 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 512 + 16 bytes spare.Boot data is protected by 4-bit ECC mode.
1'b1	1'b0	2'b01	Boot From x8 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 2K + 64 bytes spare.Boot data is protected by 4-bit ECC mode.
1'b1	1'b1	2'b01	Boot From x16 NAND flash. NAND flash is configured to 8-bits I/O bus width and a page size is 2K + 64 bytes spare.Boot data is protected by 4-bit ECC mode.

26.6.2 Booting From a NAND Flash Device

Booting from NAND flash device proceeds as follows¹:

1. BOOTLOADER copies 1 page of 4K Bytes or 1 page of 2KB or 4 pages of 1/2KB data from the NAND flash to the NFC internal RAM buffer.
2. There are five address latch cycles during the bootloader. This is hard coded and cannot be modified.
3. There is a read confirm command after the address cycles (0x30)
 - ECC is always calculated for each section (512B) separately.
 - For NAND flash with 2K page size, the first page of the NAND flash should be filled with 1/2k bytes data follow by 16 bytes of spare data 4 times as: 512 main Byte + 16 spare byte, then 512

1. A Boot from the NAND flash device only occurs if one of the Boot inputs is asserted (nf8boot_b or nf16boot_b is low) at System Power-On reset (ipp_resets rising).

- main Byte + 16spare byte, then 512 main Byte + 16spare byte, and 512 main Byte + 16spare byte
- For NAND flash with 4KB+128B page size, the first page of the NAND flash should be filled with 1/2 byte data followed by 16 bytes of spare data if 4bit ECC mode is used. As 8 times (512 main Byte + 16spare byte).
 - For NAND flash with 4KB + 218B page size, the first page of the NAND flash should be filled with 1/2 byte data followed by 26 bytes of spare data if 8bit ECC mode is used. As 7times (512 main Byte + 26spare byte) and (512 main Byte + 36 spare byte) at last.
4. The Host then reads (after exiting from reset state) the first code from the internal NAND flash controller RAMbuffer.

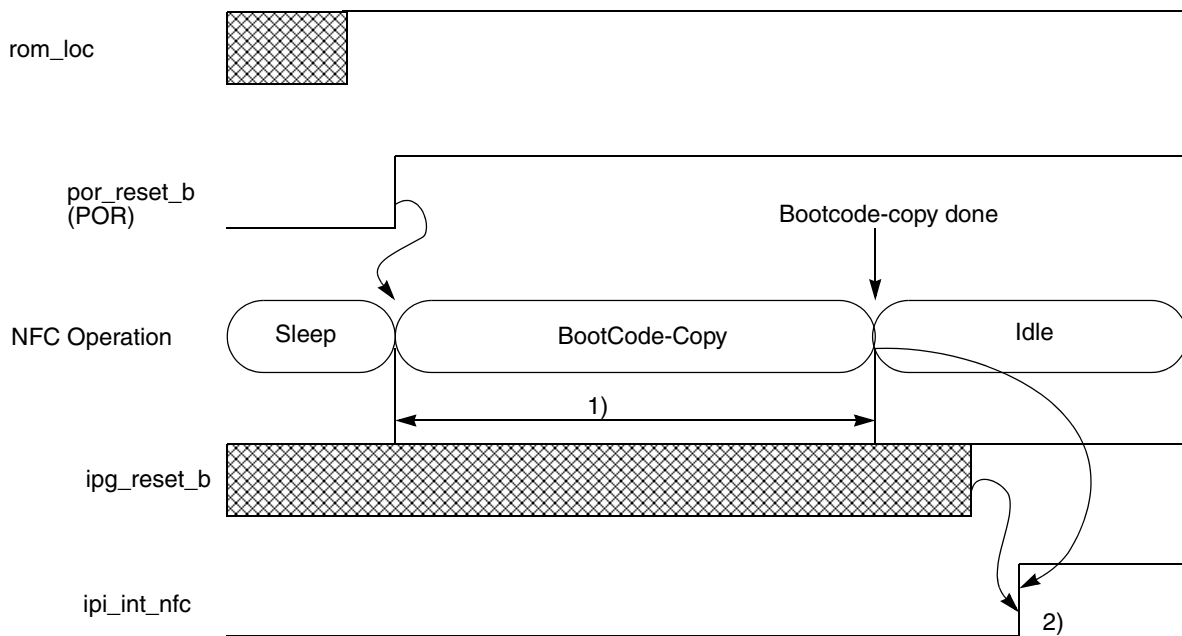


Figure 26-21. Boot Mode Operation

NOTE

The time it takes the bootcopy to load 2 K/4 K bytes is dependent on the NAND device and the frequency of flash clock. The host can read Bootcode in RAM buffer (2 K/4 K bytes) only after Bootcode copy completion.

The Interrupt pin (**ipi_int_nfc**) goes 0 to 1 when the Bootcode-copy is completed, and upon the **ipg_reset_b** rising edge. If **ipg_reset_b** goes 0 to 1 before the Bootcode-copy is done, the Interrupt pin (**ipi_int_nfc**) goes from 0 to 1 as soon as Bootcode-copy is completed.

The interrupt can be relevant for cases of secured boot (booting from ROM and then enabling the NFC boot).

26.6.3 NAND Flash Control

The NAND flash control generates all the following control signals :

- CE# (Flash Chip Enable)
- RE# (Read Enable for read operations)
- WE# (Flash Write Enable)
- CLE# (Flash Command Latch Enable), ALE (Flash Address Latch Enable).

It monitors the RB (Flash Ready/Busy indication) signal to check if the NAND flash is in the middle of operation.

The BOOTLOADER is part of the NAND flash control block.

26.6.4 NAND Flash Control

The NAND flash control generates all the control signals that control the NAND flash: nCE (flash Chip Enable), nRE (Read Enable for read operations), nWE (Flash Write Enable), CLE (Flash Command Latch Enable), ALE (Flash Address Latch Enable). It monitors the R/nB (Flash Ready/Busy indication) signal to check if the NAND flash is in the middle of operation.

The BOOTLOADER is actually a part of NAND flash Control Block.

There are several differences between 1/2k page, 2k page and 4k page size devices during program or read from NANDflash:

- 1/2 k page
Every page is written to 1/2k bytes in the nfc internal RAM buffer so it is important to change RAM buffer address register to choose to what 1/2k the page is written or read from.
After the program/read is done. RBA remains unchanged.
- 2 k page
NFC programs/reads from 4 sections of the internal RAM, so the legal values of RBA are 3'h0 or 3'h4. After the program/read is done RBA is automatically set to the next legal value (4'h0 changes to 4'h4, and vice versa)
- 4 K page
NFC programs/reads from all of the internal RAM, so the legal value of RBA is always 3'h0.

Figure 26-22 and Figure 26-23 are the example of NAND flash read. Some NAND flash need 30H command to begin read, while some needs no 30H command. See NAND flash data sheet for more detailed information.

NOTE

For boot load, NFC support only NAND flash with 512 byte page size without 30H command, as Figure 26-22 shows, or NAND flash with 2K/4k page size and support 30H command, as Figure 26-23 shows.

Figure 26-24 is NAND flash program timing diagrams. The width of row address of this example NAND flash is no more than 8 bit, so NAND flash address can be written to NAND flash in three address write operation. See NAND flash data sheet for more detailed information.

Figure 26-25 is the example of page erase timing diagrams. The width of row address of this example NAND flash is 16 bits (a9-a24). For page erase, only row address is necessary.

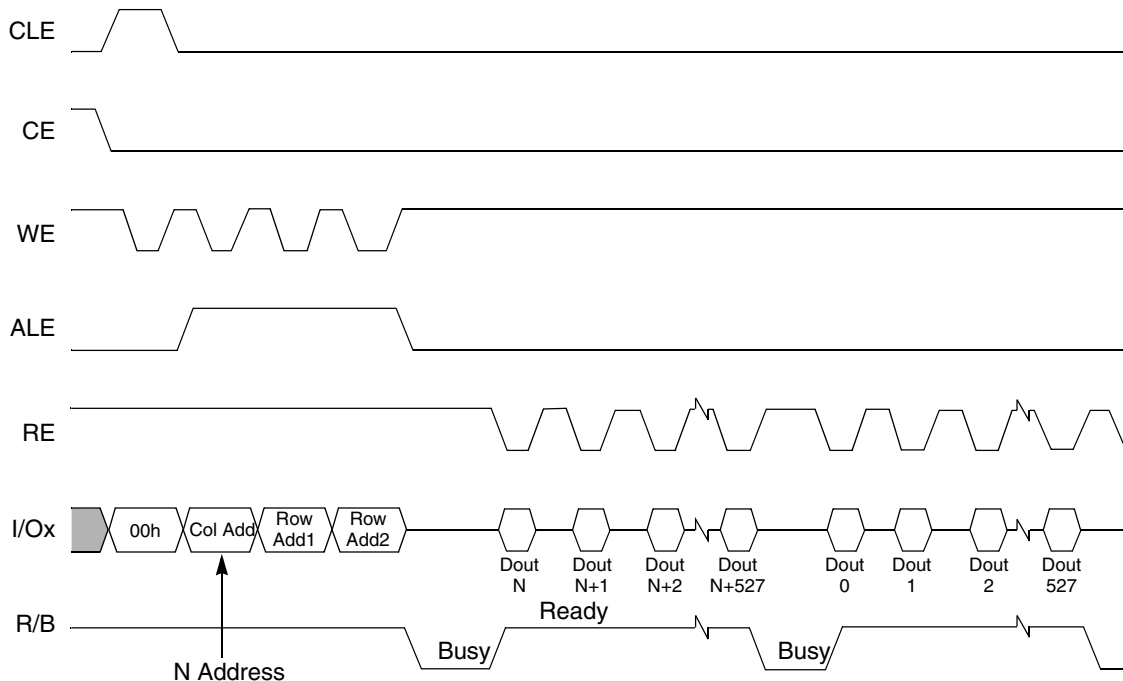


Figure 26-22. Read Operation A (Without Command 30H)

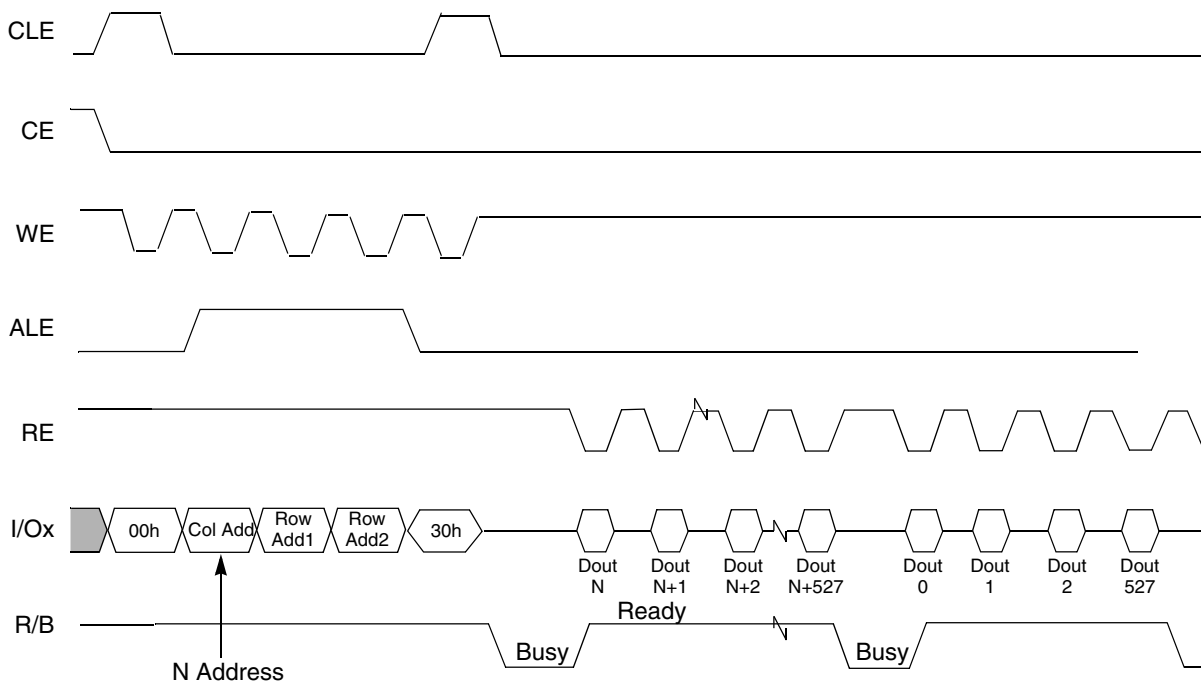


Figure 26-23. Read Operation B (With Command 30H)

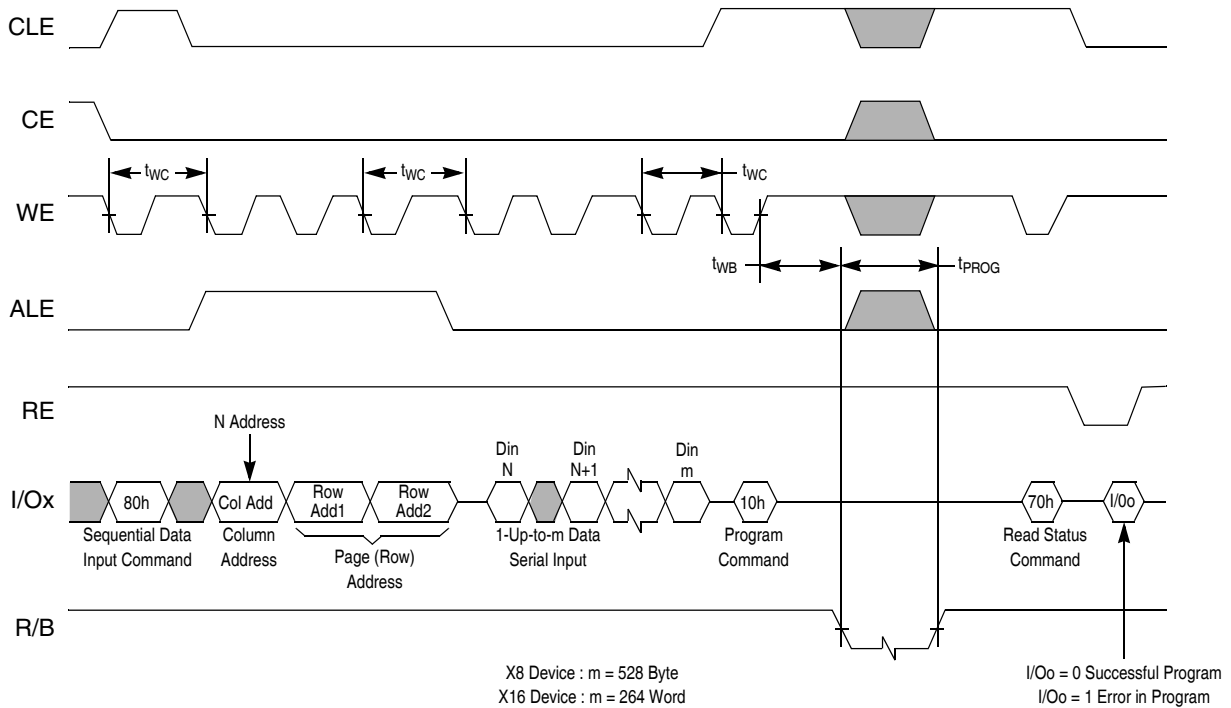


Figure 26-24. Page Program Operation

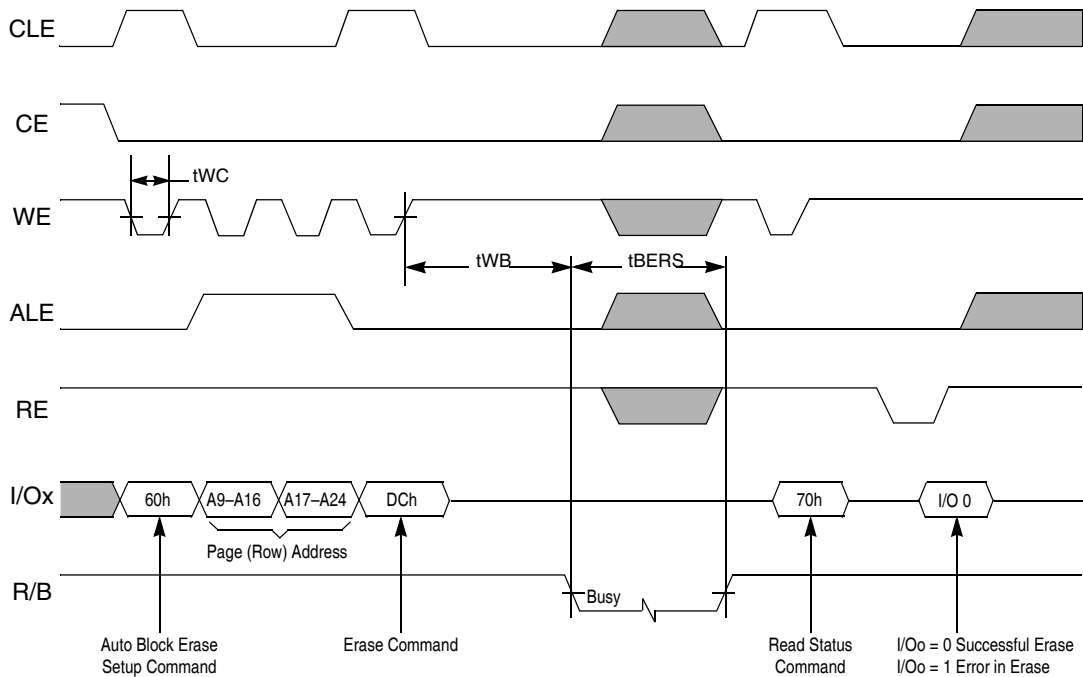


Figure 26-25. Block Erase Operation (Erase One Block)

26.6.5 Flash Clock Diagrams

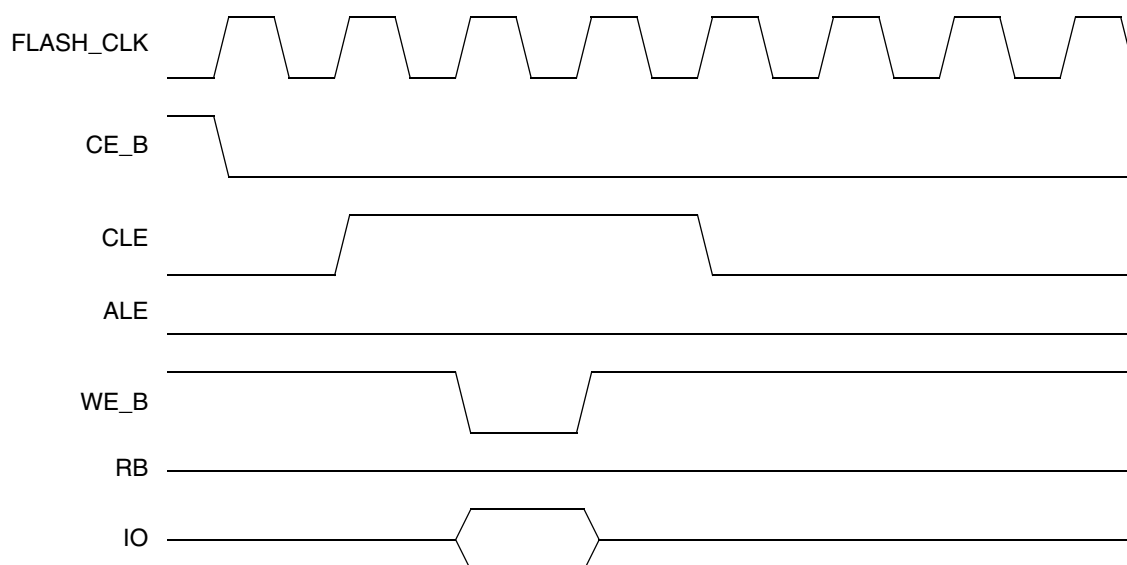


Figure 26-26. Command Latch Cycle

In [Figure 26-26](#), WE_B is in low state for 1 cycle of flash clock. CLE is in high for 3 cycles of flash clock.

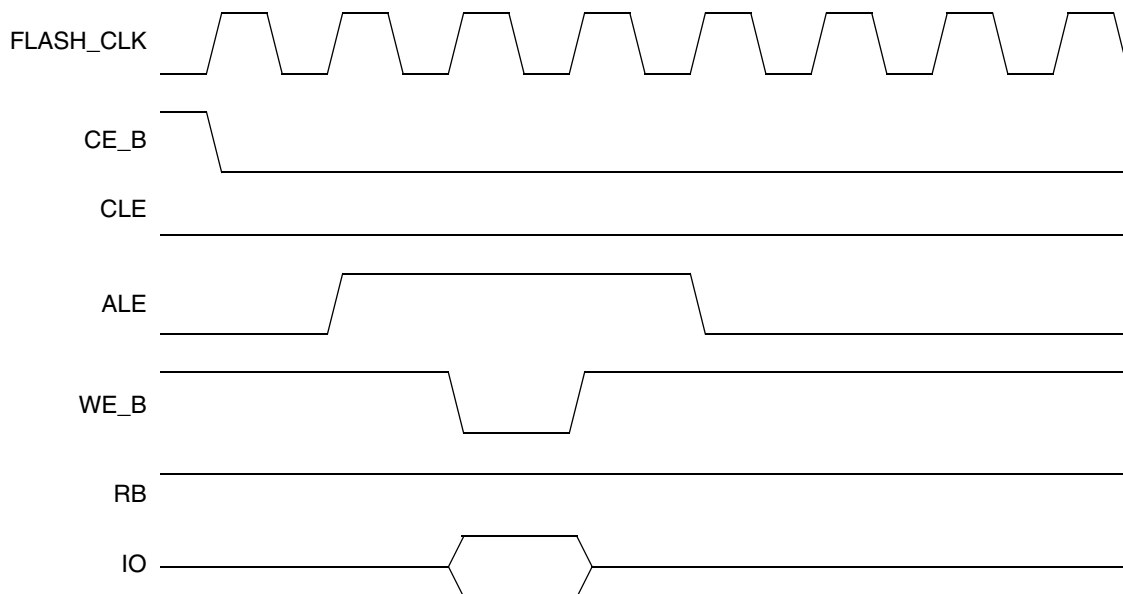


Figure 26-27. Address Latch Cycle

In [Figure 26-27](#), WE_B signal is in low state for 1 cycle of flash clock and ALE signal is in high for 3 cycles of flash.

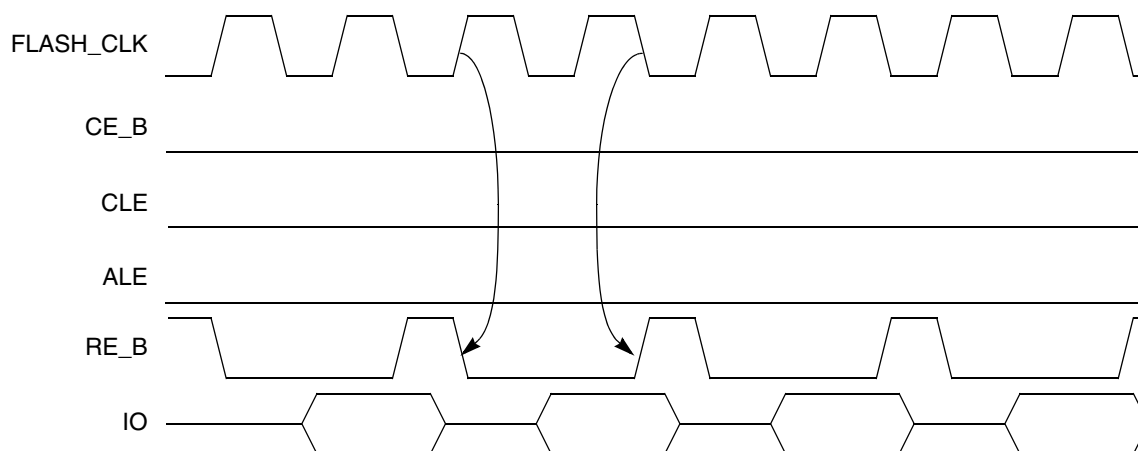


Figure 26-28. Serial Access Cycle after Read

In [Figure 26-28](#), RE_B signal is in low state for 1.5 cycles flash clock and single read access tasks 2 cycles of flash clock.

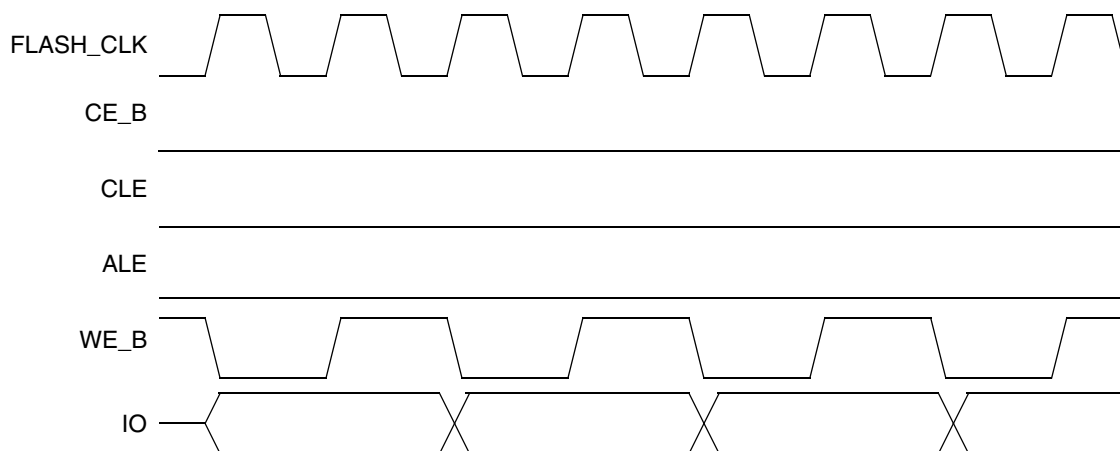


Figure 26-29. Serial Access Cycle after Write

In [Figure 26-29](#), WE_B signal is in low state for 1 cycle of flash clock. Single write access takes 2 cycles of flash clock.

26.6.6 NFC Boot Load Sequence

NFC BOOT Load logic provide two different boot load sequence for NAND flash with 512-byte page size (see [Figure 26-30](#)) and 2K/4K page size (see [Figure 26-31](#)). NFC can only boot from CE0.

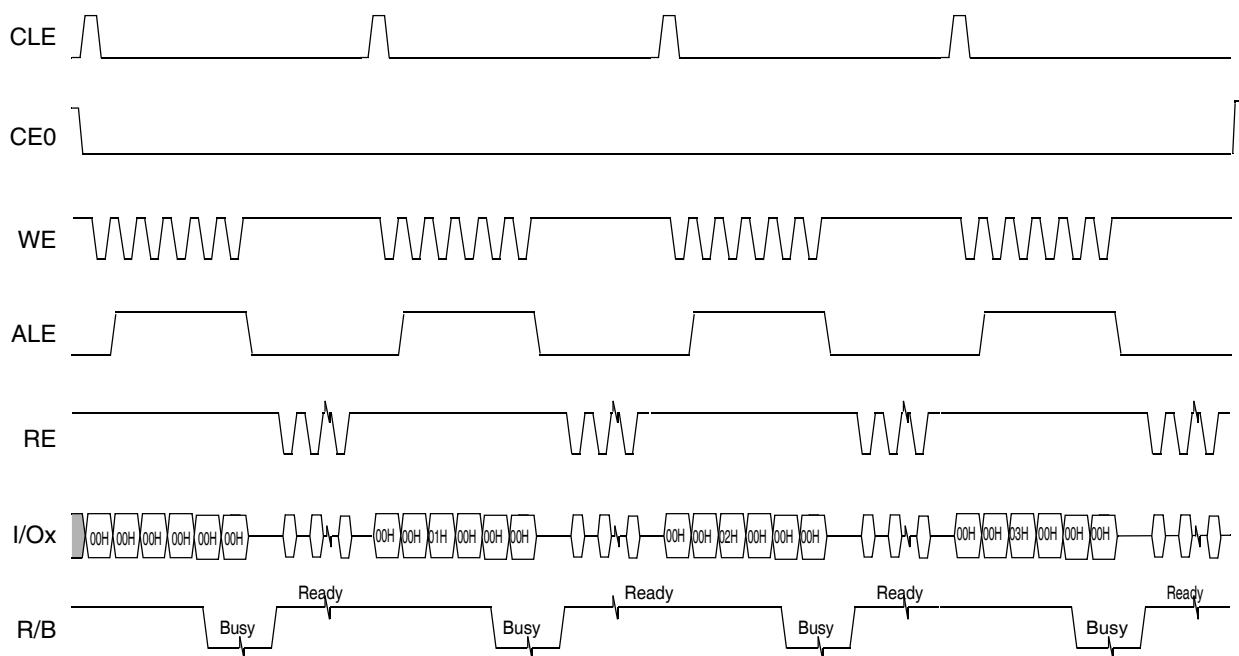


Figure 26-30. Boot sequence for NAND Flash with 512-byte page size

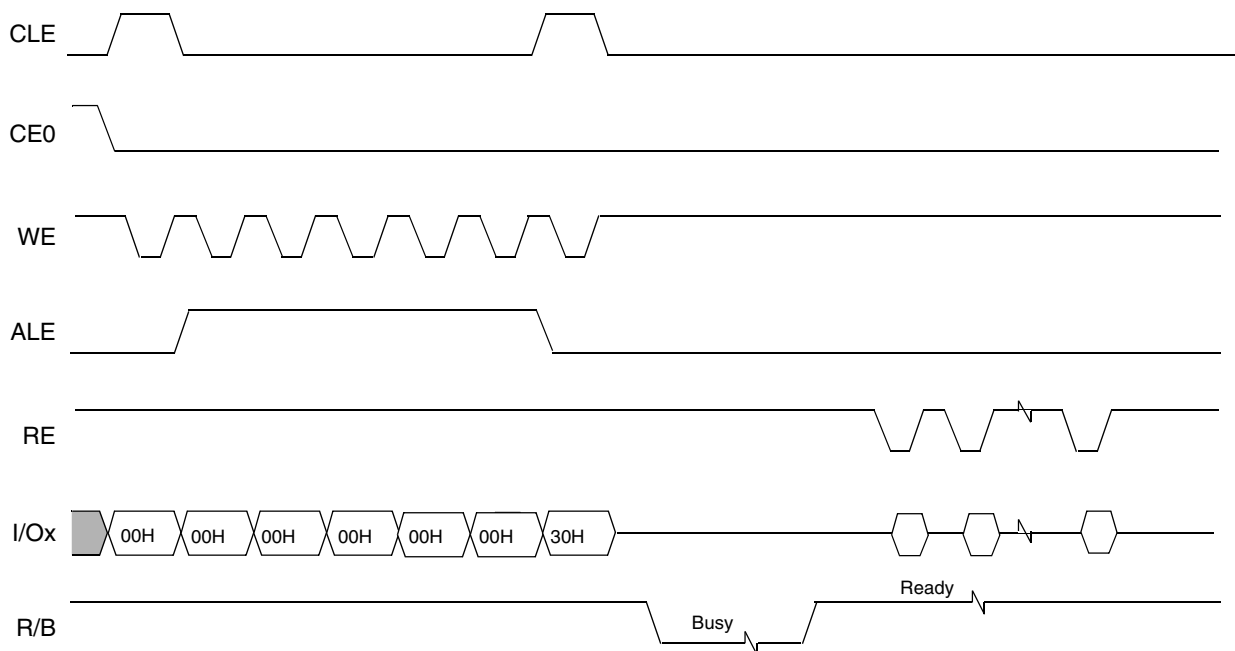


Figure 26-31. Boot sequence for NAND Flash with 2K-byte/4K-byte page size

26.6.7 DMA Request Operation

DMA request signal triggers in several cases:

- After reading a page from the NAND flash (based on dma_mode bit configuration).
- After read ID operation.

- After reading spare only area.

When asserted, the NFC asserts the dma_req signal, after CPU interface detected that address 0 of a RAM buffer is read out, dma_req is cleared automatically.

NOTE

CPU interface detects bit 5:0 of address bus only, does not care upper bits.
Software should properly set up start address of dma controller to corresponding RAM Buffer Address.

26.6.8 RS ECC

ECC engine inside NFC can correct four or eight erroneous symbols based on ecc_mode configuration bit.

For ecc_mode=0, NFC detects and correct up to eight symbols per 537B of data. You must have a device with enough spare area to use this mode (minimum required spare area size is 26Bytes per section)

For ecc_mode=1, NFC detects and correct up to four symbols per 528B of data.

The RS(511,503/495) ECC engine together with the RS ECC CONTROL blocks are responsible for detection and correction of up to four/eight symbols of 9 bits each in 528/537 byte section(main+16/25B spare).

When the NFC accesses the NAND flash device for Program operation, it generates ECC code of 9/18bytes (these bytes override the corresponding spare area data that is written in the internal RAM). When the NFC accesses the NAND flash device for a Read operation, it performs a RS detection algorithm, and indicates how many symbol errors were detected and corrected.

The ECC code is updated by the NFC automatically. After a Read operation, the host can determine whether there were errors or not by reading the status registers (ECC_Status_ResultX). The indication in the status register shows how many errors were found per 528/537 bytes.

Upon program the RS ECC bytes are written directly to the NAND flash device (and are not written back to the internal RAM). This means that if a user wants to read the ECC that was generated, he must read it from the NAND flash device spare area.

ECC is always calculated for 1/2k and not 2k/4k page even if you work with a 2K/4K page memory. In the case of 2K/4K page memory, ECC is calculated 4/8 times.

That means that the 2k/4k page in the NAND flash should be filled with 1/2k bytes data followed by spare data 4/8 times:

512 main Byte + spare bytes + 512 main Byte + spare bytes + 512 main Byte + spare bytes + ...

The ECC operation can be bypassed using ECC_EN bit in CONFIG1 register.

The way to fill the accessed NAND flash device is to perform a program sequence and a read sequence.

Program sequence:

1. Write a page of data to the internal RAM.
2. Set RBA to point to the section in which the data is in.
3. Configure the command and address sequence for program operation.
4. Program 1 page of data by writing 1 to FDI bit in config2 register (RS ECC code override the corresponding spare area bytes)

Read sequence:

1. Set RBA to point to the section in which you want the data to be written in the internal RAM.
2. Configure the command and address sequence for read operation.
3. Read 1 page by writing 1 to FDO field in config2 register (RS algorithm automatically fix up to eight error bits if there are any)

26.6.9 Address Control

This module is responsible for address control and generation. It defines the RAM buffer Address Generation (RAM buffer Address for Data In/ Data Out).

It generates and takes into account the Lock State Sequence (For more details see [Section 26.7.2.1, “Write Protection”](#)) and therefore contains the flash Memory Lock Address Comparator, and RAM buffer Lock Address Comparator which are used to determine if this area is protected or not. It also generates the RAM buffer Address for Boot Load.

26.6.10 RAM Buffer (SRAM)

The internal RAM Buffer is a 4608 Byte (4.5KB) single Port RAM buffer which is a synchronous high performance design. This memory has 1152 words of 32-bits each, from which 1024 words are used for the main buffer and the remaining 128 Words are allotted to a spare area, which is used for ECC (Error Correction) and other applications.

The NFC logically divides the RAM into eight sections of 512Bytes main data and 64Bytes of spare area. When reading (or programming) the NAND device, the NFC writes the main data from the NAND device into the main section, and the spare data from the NAND device into the spare section. If the NAND device spare-area is less than 64Bytes per 512 Bytes of main data, the NFC's spare-section in the RAM is not used fully. For example, if you use a NAND device that has 2 KB main data and 64B of spare area, this device has 16Bytes of spare for every 512Bytes of main. This spare area in the RAM is located as follows: first 16Bytes in spare section 1, next 16Bytes in spare section 2, and so on.

If you are using a NAND device in which the spare area size is not divided fully with the number of main sections, then this remainder is located at the last spare section. If you use a NAND device with 4KB page and 218Bytes of spare, it means you have 27.25 Bytes of spare for each 512Bytes of main. Then the NFC rounds this number down to the nearest even number (26), and that would be the number of bytes per section. The remaining bytes are added to the last spare section, meaning that last spare section is 36 Bytes.

This memory is used as a Bootram memory during boot from the NAND flash device, and as a buffer at normal operation.

26.6.11 Read and Write Control

The Read and Write Control Block contains a connection to the Internal bus (which is connected to the Internal RAM buffer and the registers).

It is also responsible for RAM buffer Control and Register Control, RAM buffer Lock Control and Address and Data latches.

26.6.12 Endian

This is a Big Endian system.

The endianness between the Internal RAM & external NAND flash devices is controlled by FLASH_BIG bit in CONFIG1 register. For NAND flash with 8 bit or 16 bit data width, this bit causes different data sequence in data IO. Table 26-27 shows the detailed information about the map of 32 bit IPS data and 8/16 bit of NAND flash device.

Table 26-27. ICPU data and NAND Flash data sequence

FLASH_BIG	CPU R/Wdata[31:0]	8-0bit NAND Flash Data Sequence	16-0bit NAND Flash Data Sequence
1 big-endian	AABBCCDD	0: AA 1: BB 2: CC 3: DD	0: AABB 1: CCDD
0 little-endian	AABBCCDD	0:DD 1:CC 2:BB 3:AA	0: CCDD 1: AABB

In read status operation, the location of status byte in CPU read data is shown in

Table 26-28. Status byte location

FLASH_BIG	NAND Flash Data Width	Status Byte Location in Internal RAM	Memory Map
1 big-endian	8	[0:7]	Base addr + RBA ¹ * 512
	16	[8:15]	Base addr + RBA ¹ * 512 + 1
0 little-endian	8	[24:31]	Base addr + RBA ¹ * 512 + 3
	16	[24:31]	Base addr + RBA ¹ * 512 + 3

¹ RBA is in RAM Buffer Address register, see 26.5.4.1.

26.6.13 I/O Pins Sharing

The NAND flash controller has logic that allows it to share I/O pins with pins of another memory controller.

The arbitration between the NFC & the other memory has hard priority favouring the other memory. Since NFC's accesses are long, when the other memory requests the bus, the NFC always halts its operation as soon as possible and the other memory is granted. The only operations that does not halt in the middle are short ones, such as command, address phase, or spare-area access.

This priority based arbitration mechanism must be taken into account when sharing the I/O bus with another memory. (High frequency use of the other memory, might cause NFC's accesses to be long).

26.7 Initialization Information

This section describes how to operate the NFC using its registers and its interrupts, and is divided into the following subjects:

- Normal operation – To operate a NAND flash device using the NFC, use the instructions in the section Normal Operation ([26.7.1/26-34](#)).
- ECC operation – ECC operation is used when an error is detected.
- Write protection operation (to the internal memory and the flash device) – Write protection is used when the programmer wishes to protect part of the NAND flash device memory from being written except in certain cases. There are two levels of protection: software (for frequently-changed memory locations), and hardware (for memory locations whose contents are rarely changed).

26.7.1 Normal Operation

Normal operation is composed of fundamental building block operations (in [26.7.1.1/26-35](#)), in addition to specific operations, as shown in the flow charts below (in Sections [26.7.1.2.1/26-40](#) to [26.7.1.2.6/26-47](#)).

26.7.1.1 Fundamental Building Block Operations

26.7.1.1.1 Preset Operation

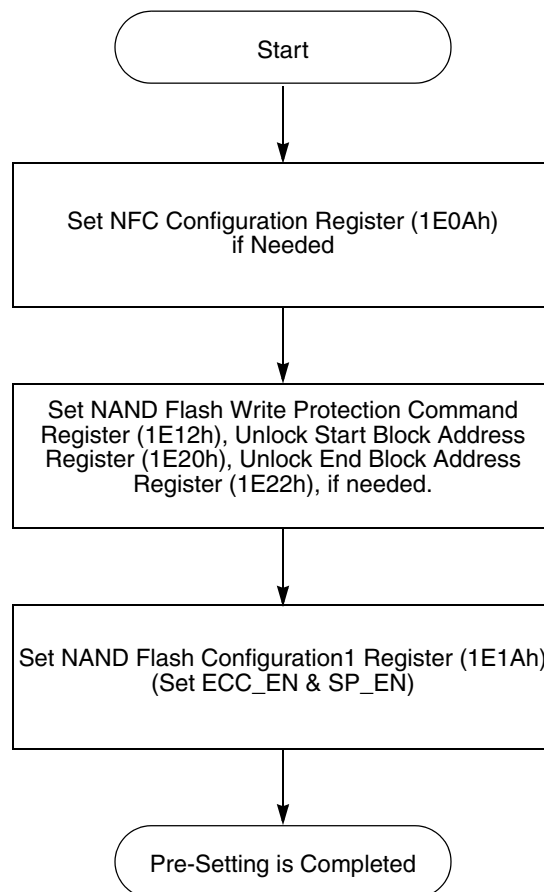


Figure 26-32. Flow Chat of Preset Operation

26.7.1.1.2 NAND Flash Command Input Operation

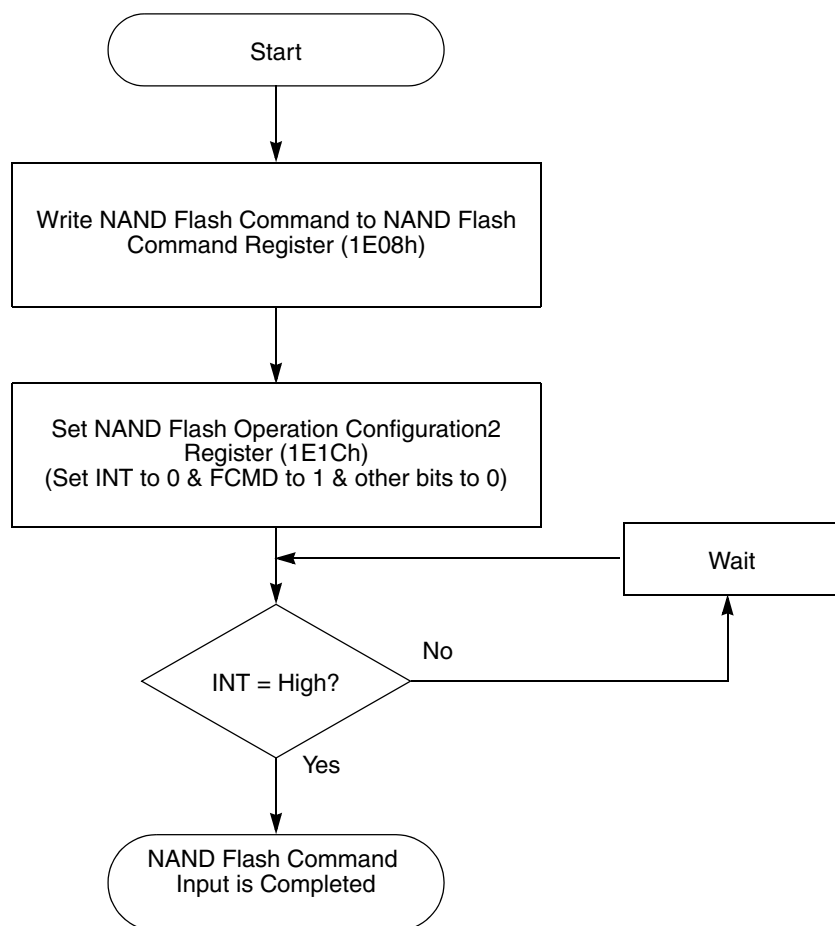


Figure 26-33. Flow Chart of NAND Flash Command Input Operation

26.7.1.1.3 NAND Flash Address Input Operation

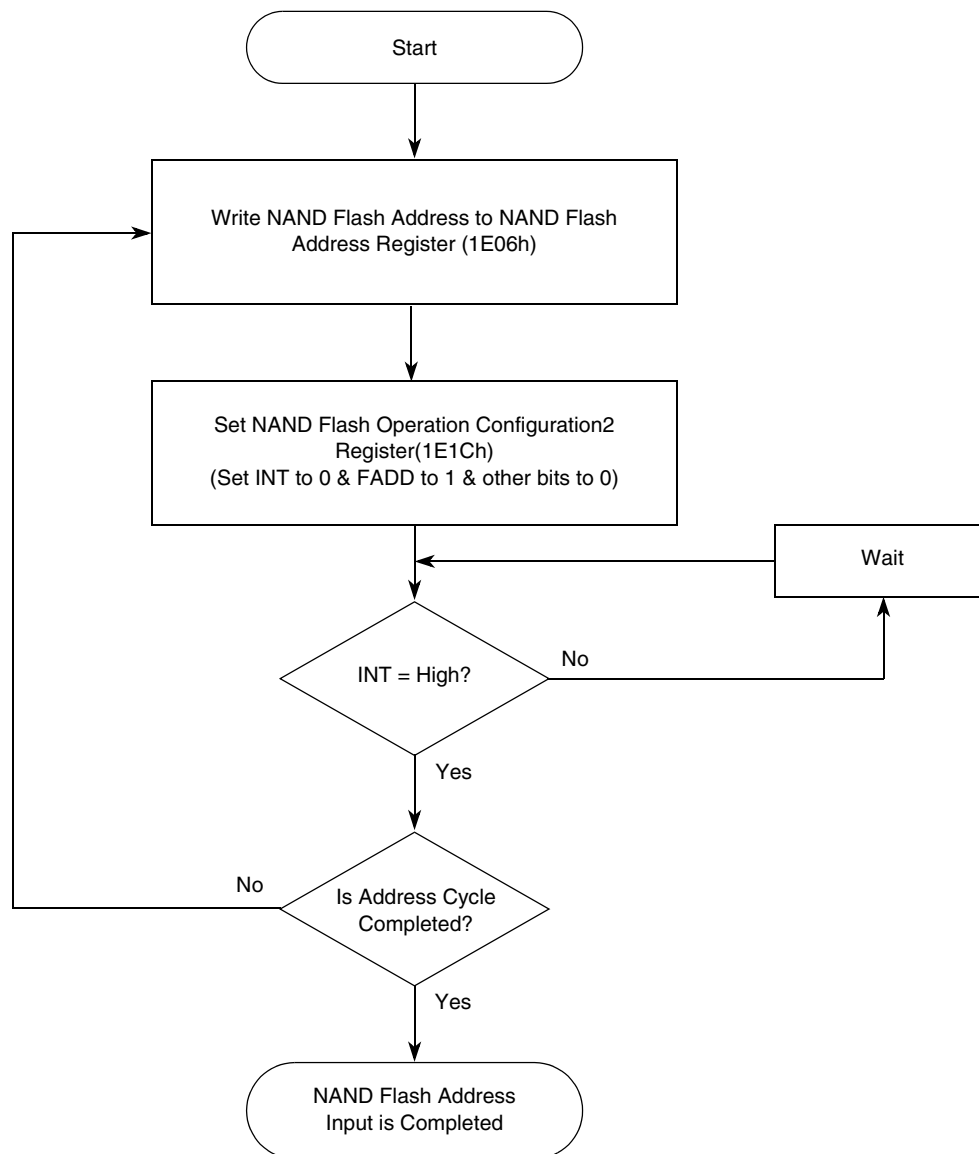


Figure 26-34. Flow Chart of NAND Flash Address Input Operation

26.7.1.1.4 NAND Flash Data Input (Program) Operation

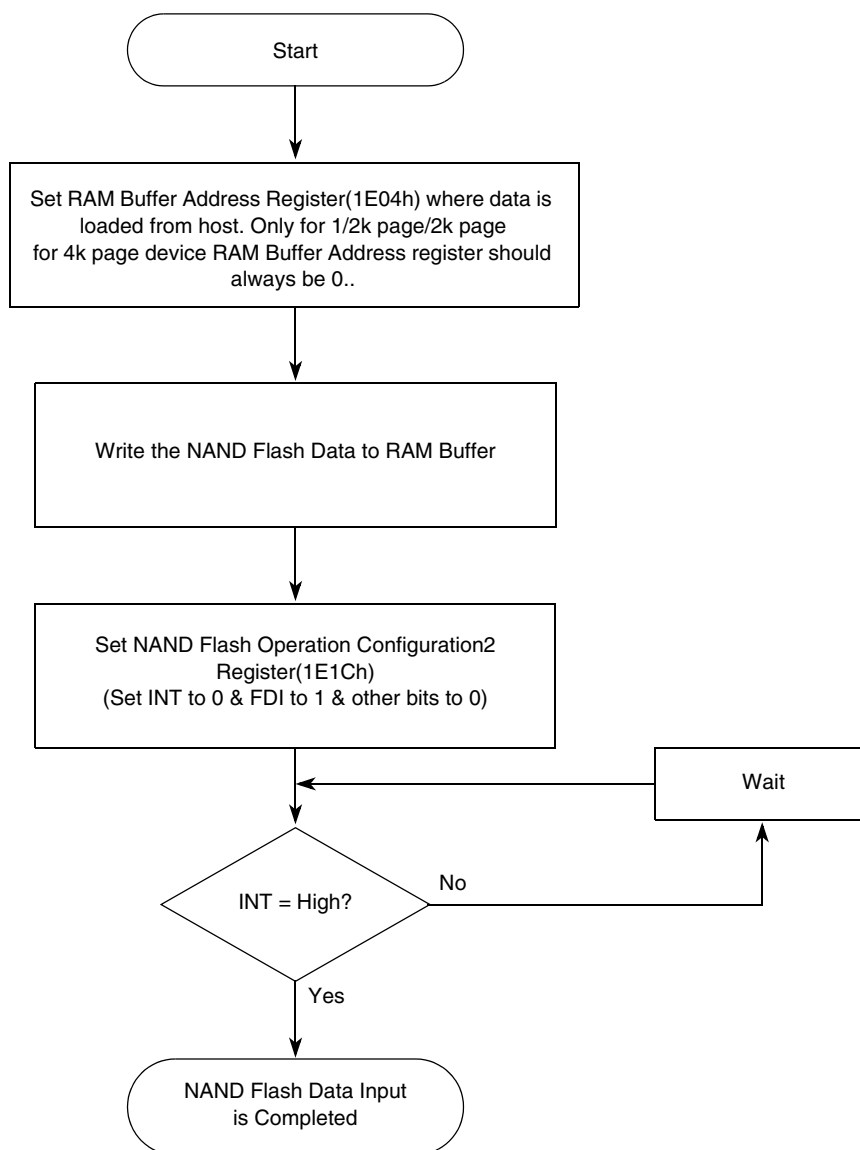


Figure 26-35. Flow Chart of NAND Flash Data Input Operation

26.7.1.1.5 NAND Flash Data Output Operation (Read)

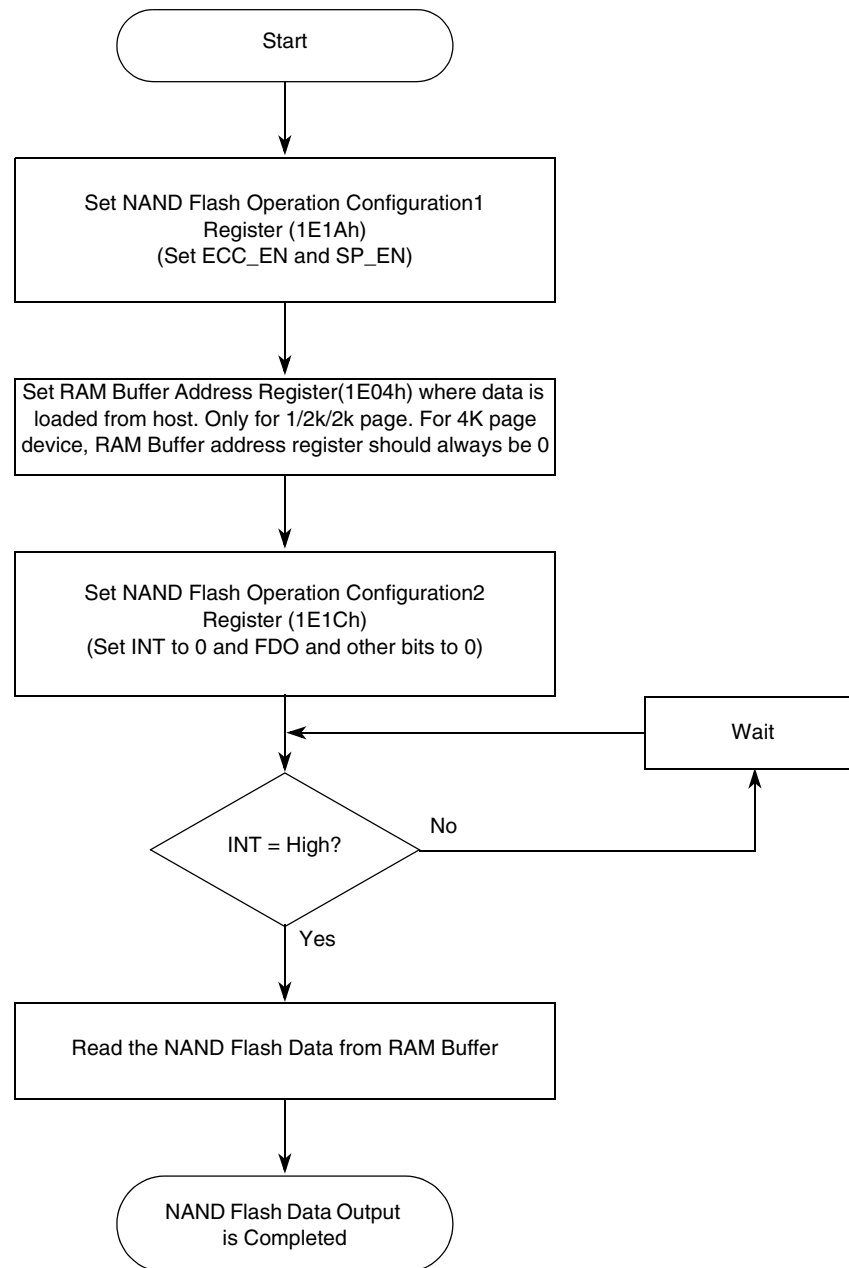


Figure 26-36. Flow Chart of NAND Flash Data Output Operation

26.7.1.2 Normal Operation

Normal operations are composed of basic operations.

26.7.1.2.1 Read NAND Flash ID Read Operation

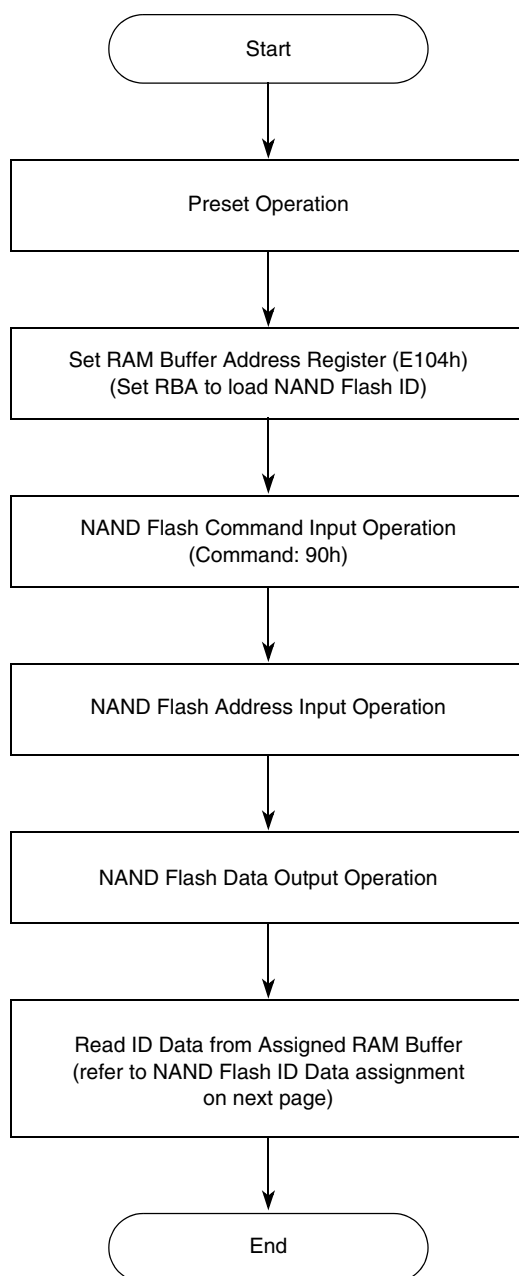


Figure 26-37. Flow Chart of Read NAND Flash ID Operation

The assignment of NAND flash ID data stored in RAM buffer (In case of X8 org. NAND flash).

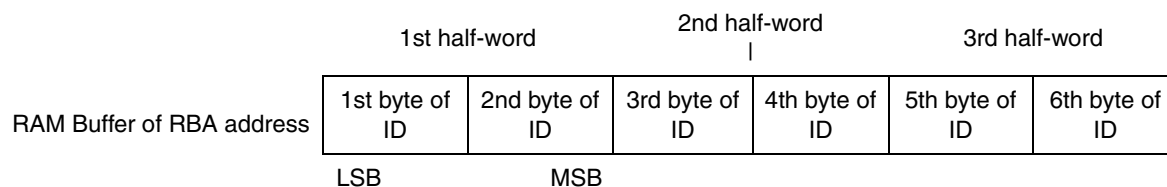


Figure 26-38. X8 Assignment of NAND Flash ID Data

The assignment of NAND flash ID data stored in RAM buffer (In case of X16 org. NAND flash).

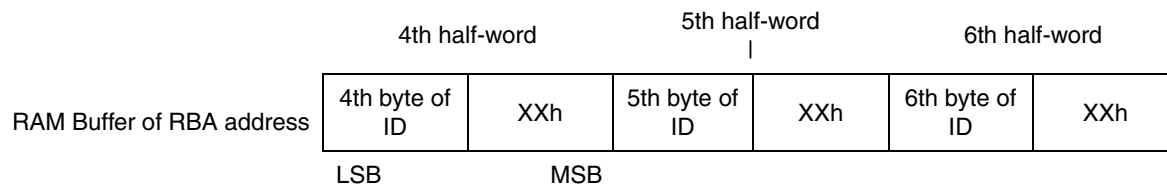
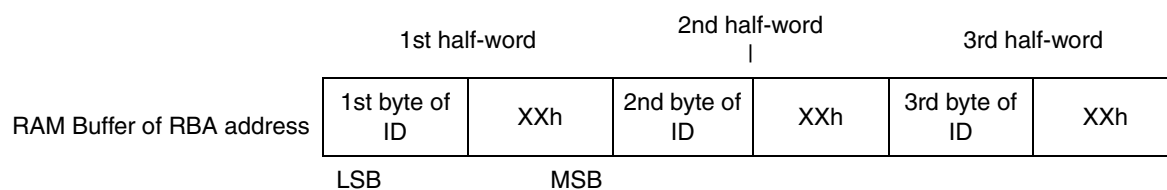


Figure 26-39. X16 Assignment of NAND Flash ID Data

26.7.1.2.2 NAND Flash Status Read Operation

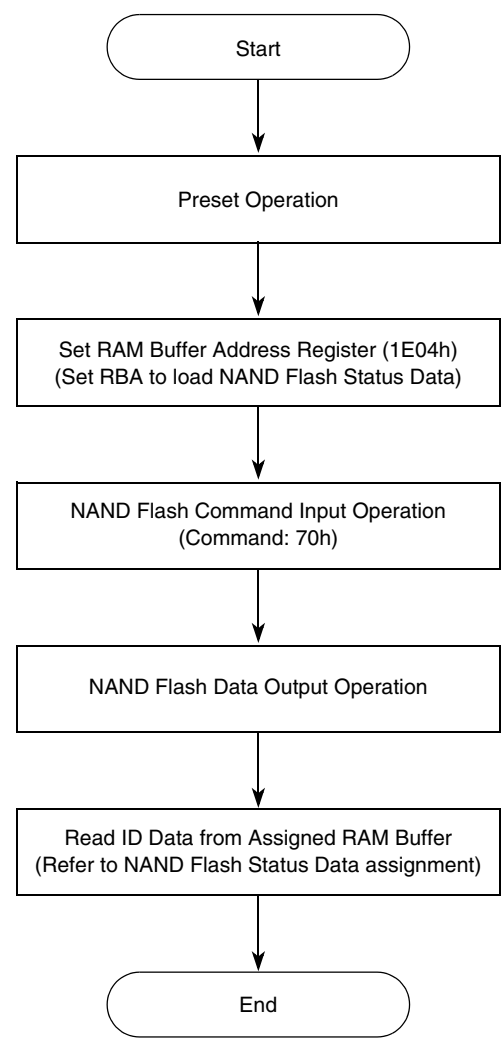


Figure 26-40. Flow Chart of Read NAND Flash Status Operation

Table shows the assignment of NAND flash status data stored in RAM buffer in little endian mode (In case of X8/X16 org. See Section 26.5.4.10, “NAND Flash Operation Configuration (NF_CFG1) for information about endian mode)

Table 26-29. Status byte in little endian mode

RAM Buffer of RBA address	1st word			
	XXh	XXh	XXh	Status Byte
	LSB			MSB

Table shows the assignment of NAND flash status data stored in RAM buffer in big endian mode (In case of X8 org. See Section 26.5.4.10, “NAND Flash Operation Configuration (NF_CFG1) for information about endian mode)

Table 26-30. Status byte in big endian mode (x8 org)

RAM Buffer of RBA address	1st word			
	Status Byte	XXh	XXh	XXh
	LSB			MSB

[Table](#) shows the assignment of NAND flash status data stored in RAM buffer in big endian mode (In case of X16 org. See [Section 26.5.4.10, “NAND Flash Operation Configuration \(NF_CFG1\)”](#) for information about endian mode)

Table 26-31. Status byte in big endian mode (x8 org)

RAM Buffer of RBA address	1st word			
	XXh	Status Byte	XXh	XXh
	LSB			MSB

26.7.1.2.3 Read NAND Flash Data Operation

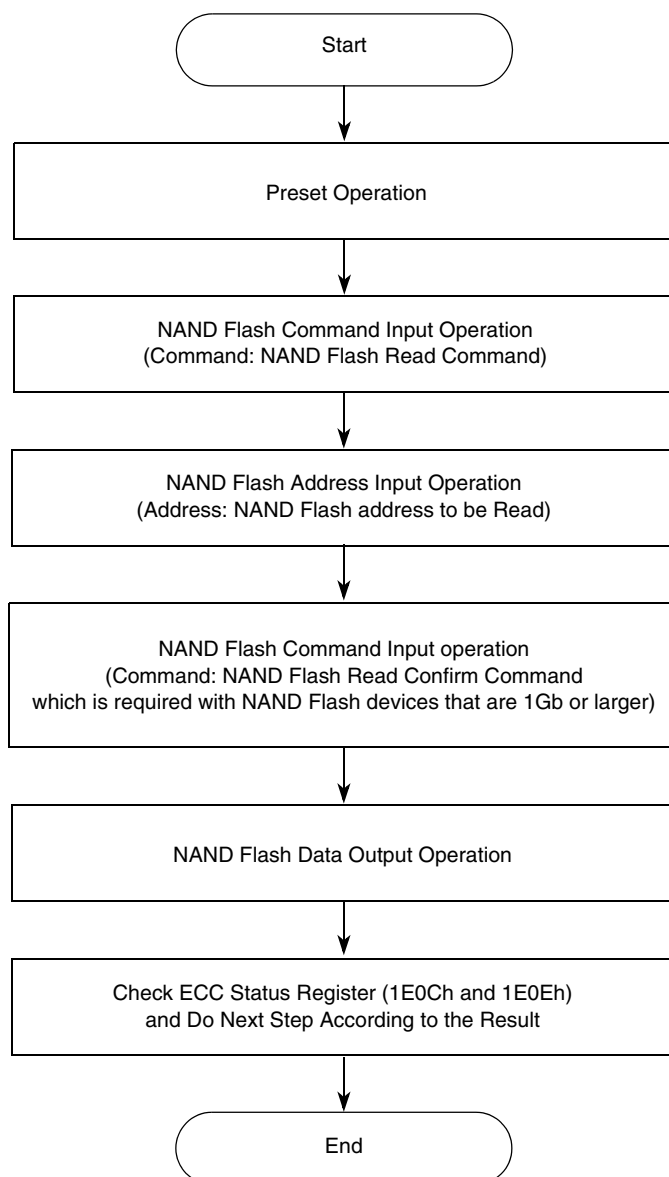


Figure 26-41. Flow Chart of Read NAND Flash Data Operation

26.7.1.2.4 Program NAND Flash Data Operation

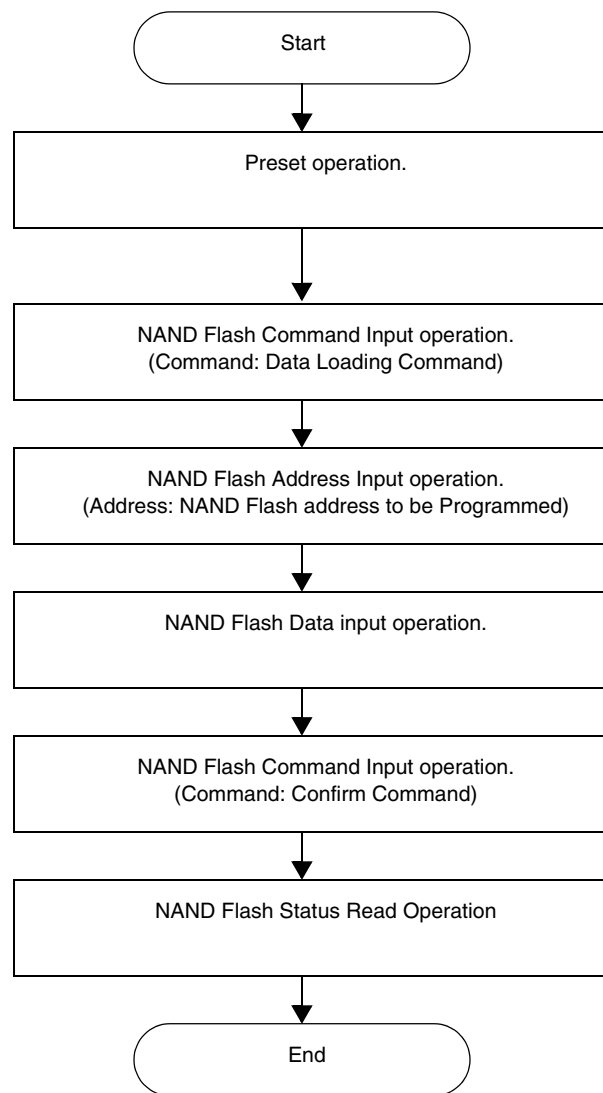


Figure 26-42. Flow Chart of Program NAND Flash Data Operation

26.7.1.2.5 Erase NAND Flash Data Operation

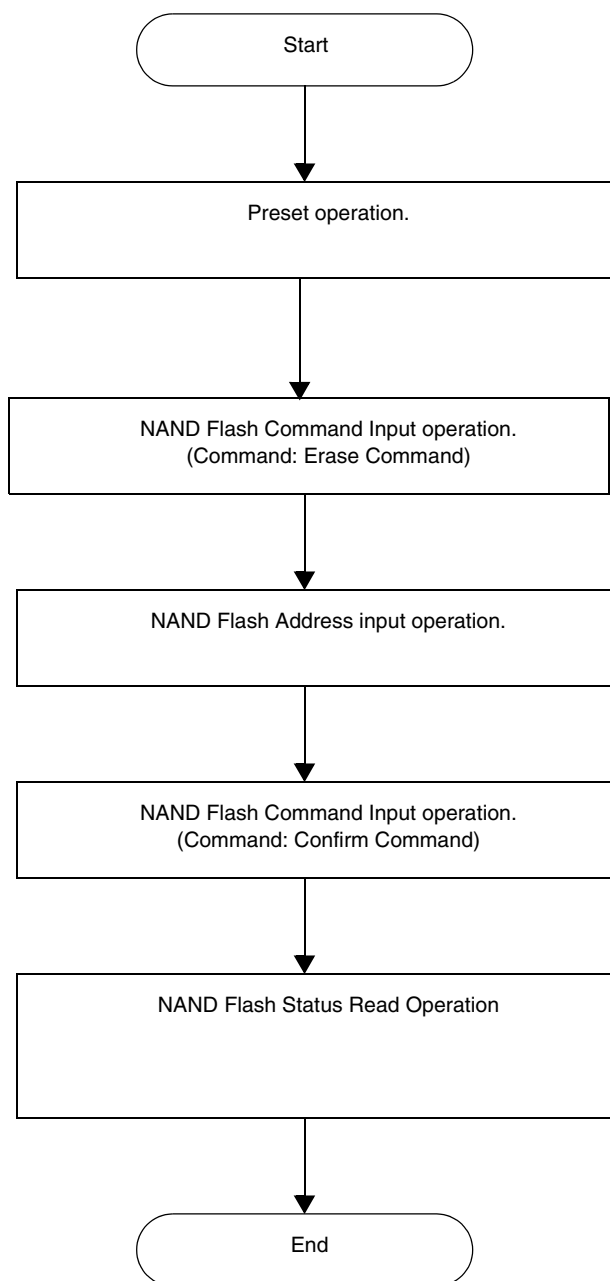


Figure 26-43. Flow Chart of Erase NAND Flash Operation

26.7.1.2.6 HOT Reset (Controller and NAND Flash Reset)

Hot reset makes controller and NAND flash stop current operation and internal registers go to default state.

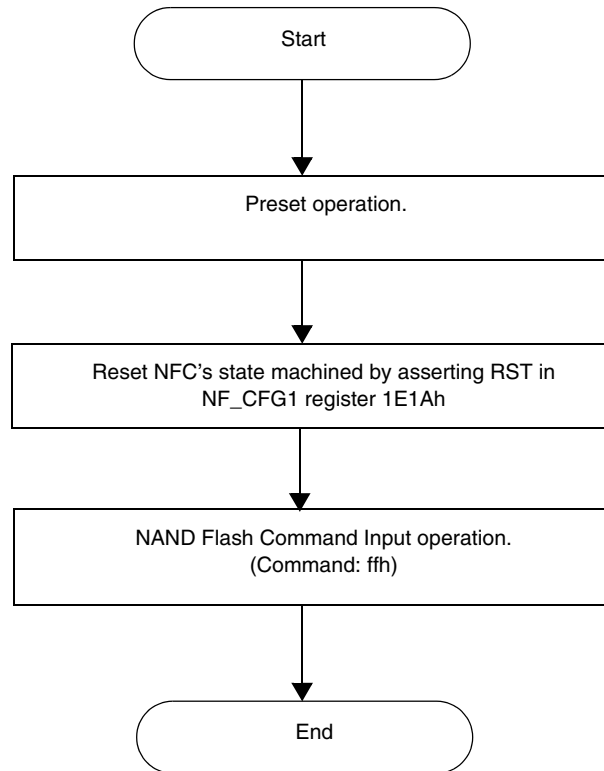


Figure 26-44. Flow Chart of Hot Reset Operation

26.7.1.3 ECC Operation

26.7.1.3.1 ECC Normal Operation

When the NFC accesses the NAND flash device for Program operation, it generates ECC code (9/18 bytes for each 528/538 bytes).

When the NFC accesses the NAND flash device for a Read operation, it reads the ECC code, detects the number of errors and their position and corrects up to four/eight symbols (9 bits each symbol) if applicable. [Table 26-32](#) shows the ECC code assignment of the NAND flash spare area. This ECC code is updated by NFC automatically.

After the Read operation, the host can know whether there are errors or not by reading the status registers (see the ECC_STATUS1/ ECC_STATUS2 register).

Since the generated ECC code is not updated to the internal buffer RAM, but is updated to the NAND flash spare area immediately upon program operation, the host can read generated ECC code only from NAND flash spare area.

26.7.1.4 ECC Bypass operation

In ECC bypass operation, the spare area is copied from NFC internal RAM Buffer to the NAND flash device during program operation. During read operation, the ECC detect-fix mechanism does not work and the ECC status register is not updated.

Table 26-32. ECC Code/Result Readability

Operation	Read operation		Program operation		
	ECC Code from spare area buffer	ECC status register	ECC Code from spare area buffer	ECC status register	ECC content in NAND flash device
ECC operation	ECC code copied from NANDflash device spare area	Valid	Invalid (old data ¹)	—	ECC code generated by the NFC
ECC bypass	User data copied from NANDflash device spare area	Not valid	Invalid (old data)	—	Data from spare area of NFC's internal RAM

¹ Old data: ECC code is not updated to spare buffer, so ECC code placement of spare buffer remains old data.

26.7.1.4.1 ECC Operation Guidance

ECC generation and correction by NFC: Program with ECC operation/read with ECC operation.

ECC generation by NFC and correction by Host: Program with ECC operation/read without ECC operation.

26.7.2 Symmetric Mode – One Flash Clock Cycle Per Input or Output Data Cycle

In the default state of the NFC two flash clock cycles are used for one access of RE#/WE#.

To not work with high frequency of flash clock, the SYM bit in config1 register should be set and it changes the WE# and RE# period during read or program to be 1 flash clock instead of 2.

SYM bit also changes the duty cycle of RE# to be ~50% during read operation.

When using SYMMETRIC mode, the data is latched into NFC on falling edge of RE#. When not using SYMMETRIC mode, the data is latched on rising edge of RE#.

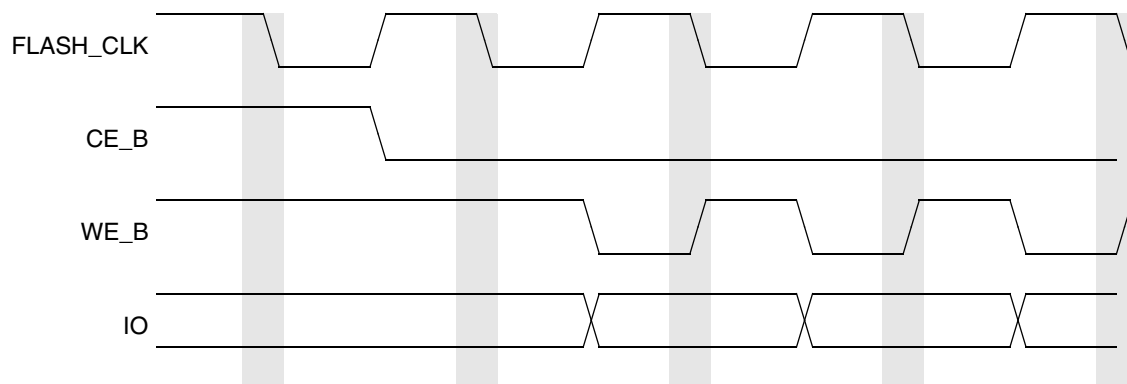


Figure 26-45. One Flash Clock Cycle Per Data Input (SYM bit = 1)

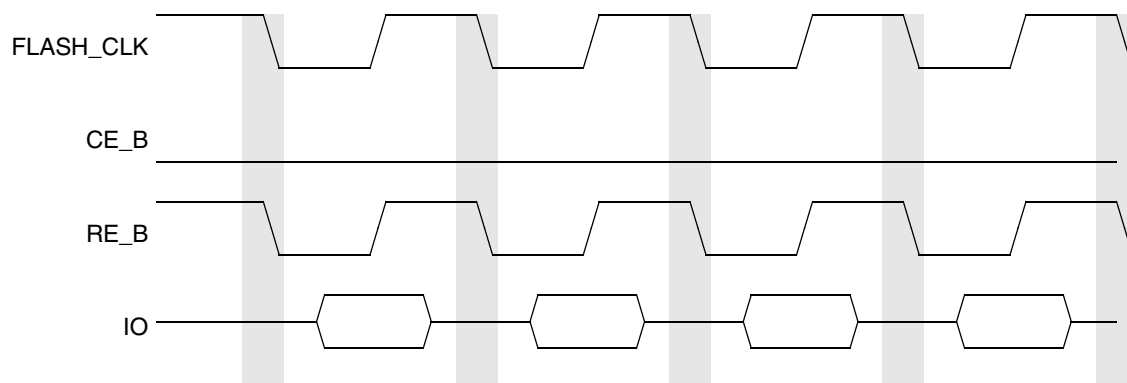


Figure 26-46. One Flash Clock Cycle Per Data Output (SYM bit = 1)

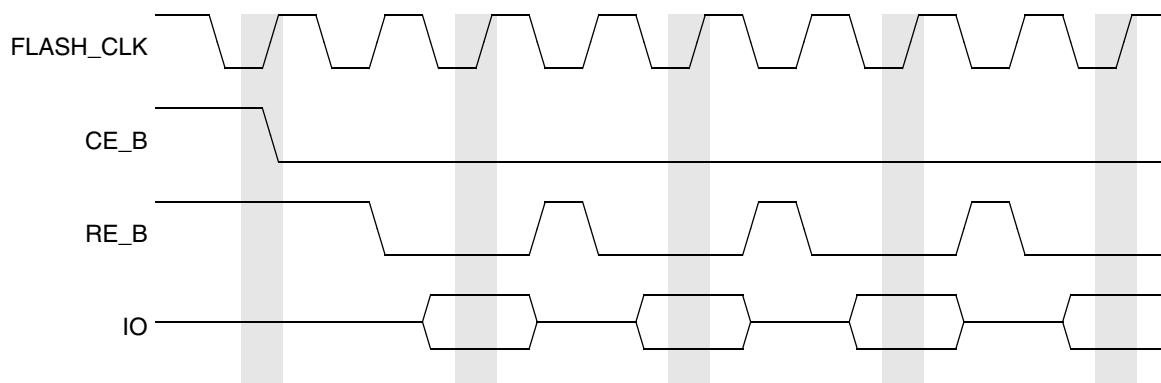


Figure 26-47. Two Flash Clock Cycles Per Data Output (SYM bit = 0)

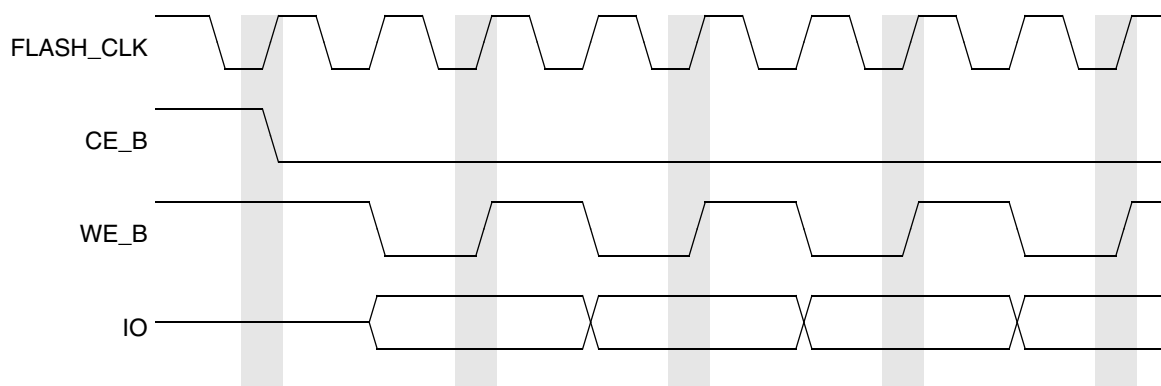


Figure 26-48. Two Flash Clock Cycles Per Data Input(SYM bit = 0)

26.7.2.1 Write Protection

The NFC offers a software write protection feature, and a hardware write protection feature. Both are described in this section.

26.7.2.1.1 WRITE Protection for RAMbuffer (LSB 2KB)

The NFC offers a software write protection feature for the first 2KB (+ accompanied spare area data) of the RAM buffer, which protects RAM buffer data. This write protection is carried out by setting the BLS bit of the NFC_CONFIGURATION register.

The default state is locked state, and the first 2KB go to this state after a cold or warm reset.

Write protection availability for main/spare memory regions in the RAM buffer are described on [Table 26-33](#). A state diagram of RAM buffer write protection is shown in [Figure 26-49](#)

Table 26-33. Write Protection for Main/Spare RAM Buffer

Main area	Spare area	
First section of RAMbuffer	1st section of RAMbuffer	Write Protection Available
Second section of RAMbuffer	2nd section of RAMbuffer	
Third section of RAMbuffer	3rd section of RAMbuffer	
Fourth section of RAMbuffer	4th section of RAMbuffer	
Fifth section of RAMbuffer	5th section of RAMbuffer	Write Protection not available
Sixth section of RAMbuffer	6th section of RAMbuffer	
Seventh section of RAMbuffer	7th section of RAMbuffer	
Eighth section of RAMbuffer	8th section of RAMbuffer	

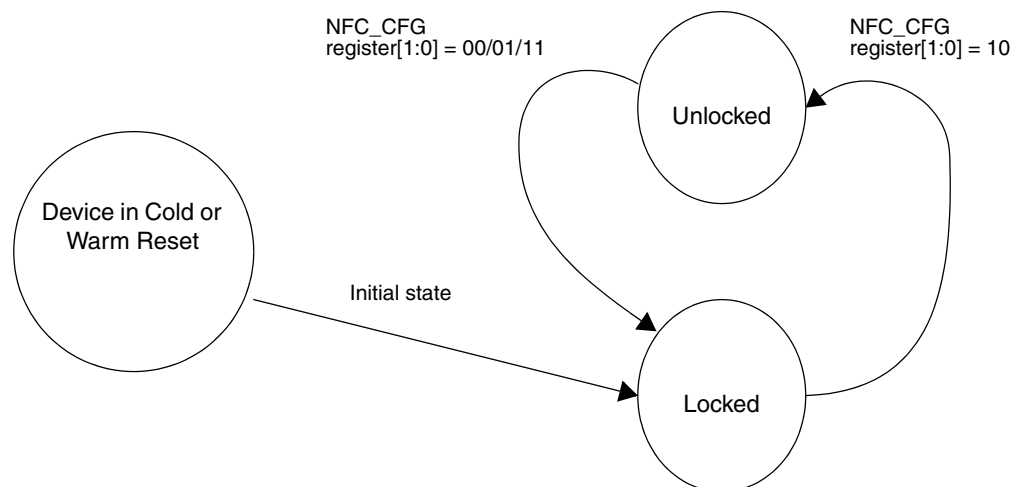


Figure 26-49. State Diagram of RAMbuffer Write Protection

26.7.2.1.2 Write Protection Modes

NFC offers hardware and software Write Protection options for the NAND flash device. The software Write Protection feature is used by executing the Lock block command or lock-tight block command, and the hardware Write Protection feature is used by executing a cold or warm reset. The WP signal is asserted only upon POR.

26.7.2.1.3 Write Protection Commands

There are two write protection states, locked and lock-tight.

- Locked state means that memory block in question is write protected (it cannot be written to), but the `UNLOCK` command can unlock it. Useful for frequently changed memory blocks.
- Lock-Tight state is a higher level of protection, and means that the memory block in question is write protected, but the `UNLOCK` command cannot unlock it. Useful for memory blocks whose contents are rarely changed.

The followings summarizes the locking functionality.

- All blocks power-up in a locked state except block zero. The unlock command can unlock these blocks.
- The Lock-tight block command locks blocks and prevents it (them) from being unlocked.
 - Lock-Tight state can be reverted to locked state only when Cold/Warm reset is executed.
- Writing to the unlock start/end address registers (`Unlock_Start_Blk_Add` and `Unlock_End_Blk_Add`) while the NFC is in the Lock-Tight state does not affect the unlock address.

26.7.2.1.4 Write Protection Status

The current Write Protection status of the NFC can be read in NAND flash write protection status register (`NAND_Flash_WR_Pr_St`). There are three bits: `US`, `LS`, and `LTS`, which are not cleared by hot reset. These Write Protection status bits are updated only when a command is issued to the NAND device.

Figure 26-50 shows a state diagram for the write protection of the NFC.

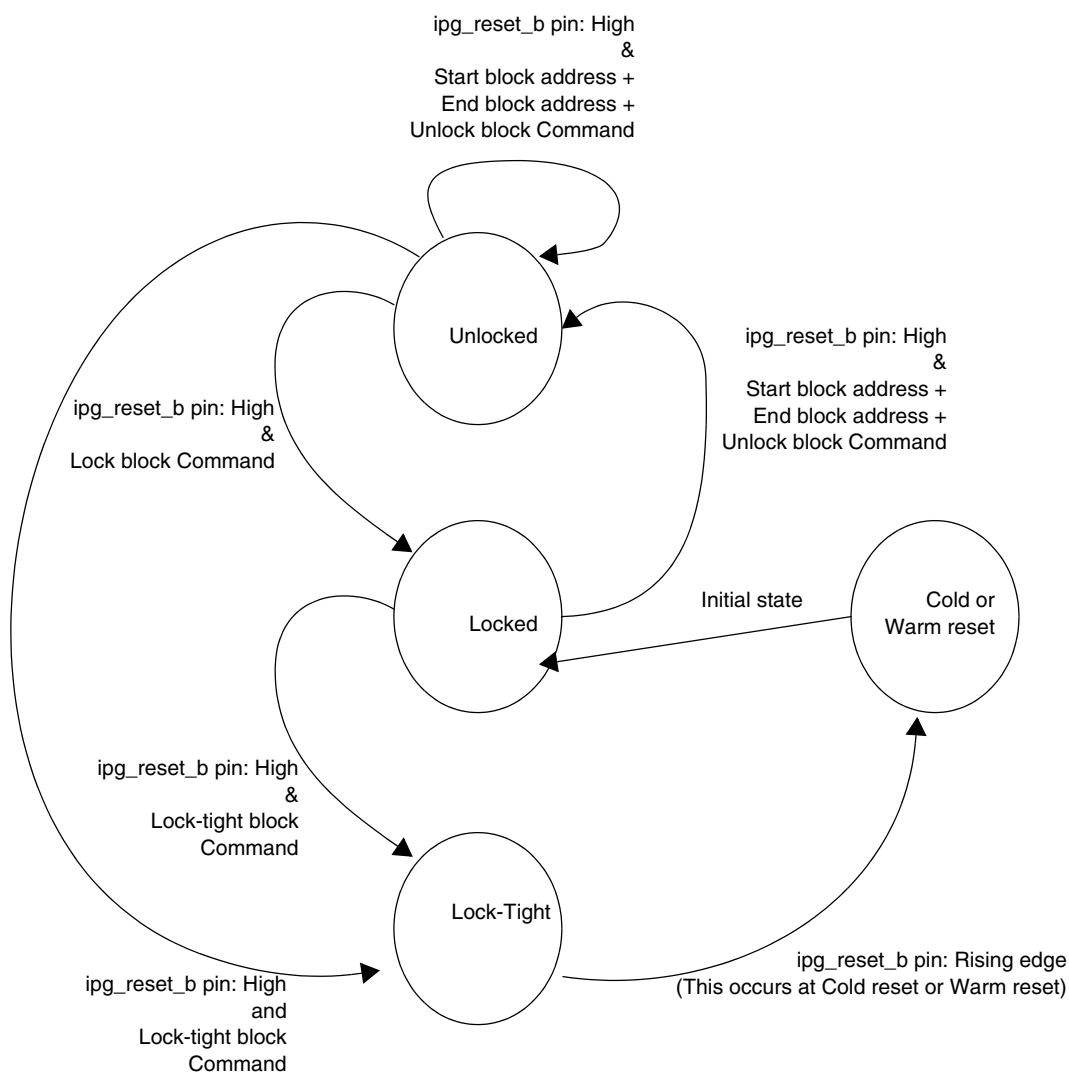


Figure 26-50. State Diagram of NAND Flash Write Protection

26.7.2.1.5 Lock Sequence

The following describes the lock sequence:

1. Command Sequence: Lock block Command (02h)
2. All blocks default to locked after initial Cold reset or Warm reset
3. Locking some of the blocks is not available; all memory blocks are locked upon reset except block zero.
4. Unlocked memory blocks can be locked by using the Lock block command. The status of a locked memory block can be changed to unlocked or lock-tight using the appropriate software commands.

26.7.2.1.6 Unlock Sequence

The following describes the unlock sequence:

1. Command Sequence: Start block address + End block address + Unlock block Command(04h)
2. Unlocked blocks can be programmed or erased
3. The status of an unlocked block can be changed to locked or lock-tight using the appropriate software commands
4. Only one successive area can be released to unlocked state from locked state; Unlocking multi areas is not available

0.0.0.0.1 Lock-tight Sequence

The following describes the lock-tight sequence:

1. Only locked blocks can be locked-tight by the lock-tight block command.
2. Command Sequence: Lock-tight block Command (01h)
3. Unlocking multi area is not available
4. Lock-tight blocks revert to the locked state at Cold/Warm reset.

26.7.3 Memory Configuration Examples

The following figures show memory connection for various bit values: An 8-bit configuration example is shown in [Figure 26-51](#), a 16-bit configuration example is shown in [Figure 26-52](#).

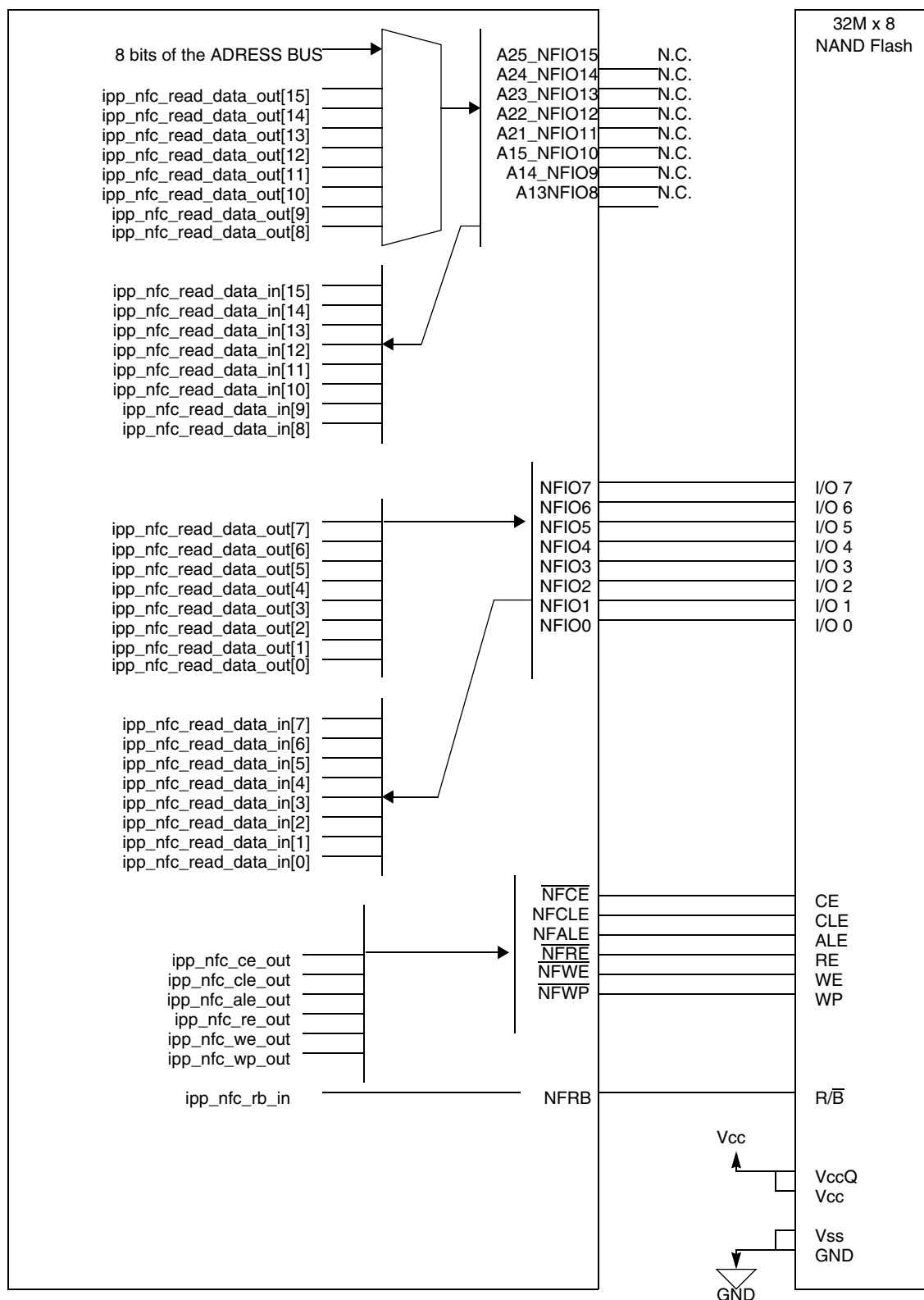


Figure 26-51. 256 Mbit (32M x 8 Bit) NAND Flash Connection Diagram

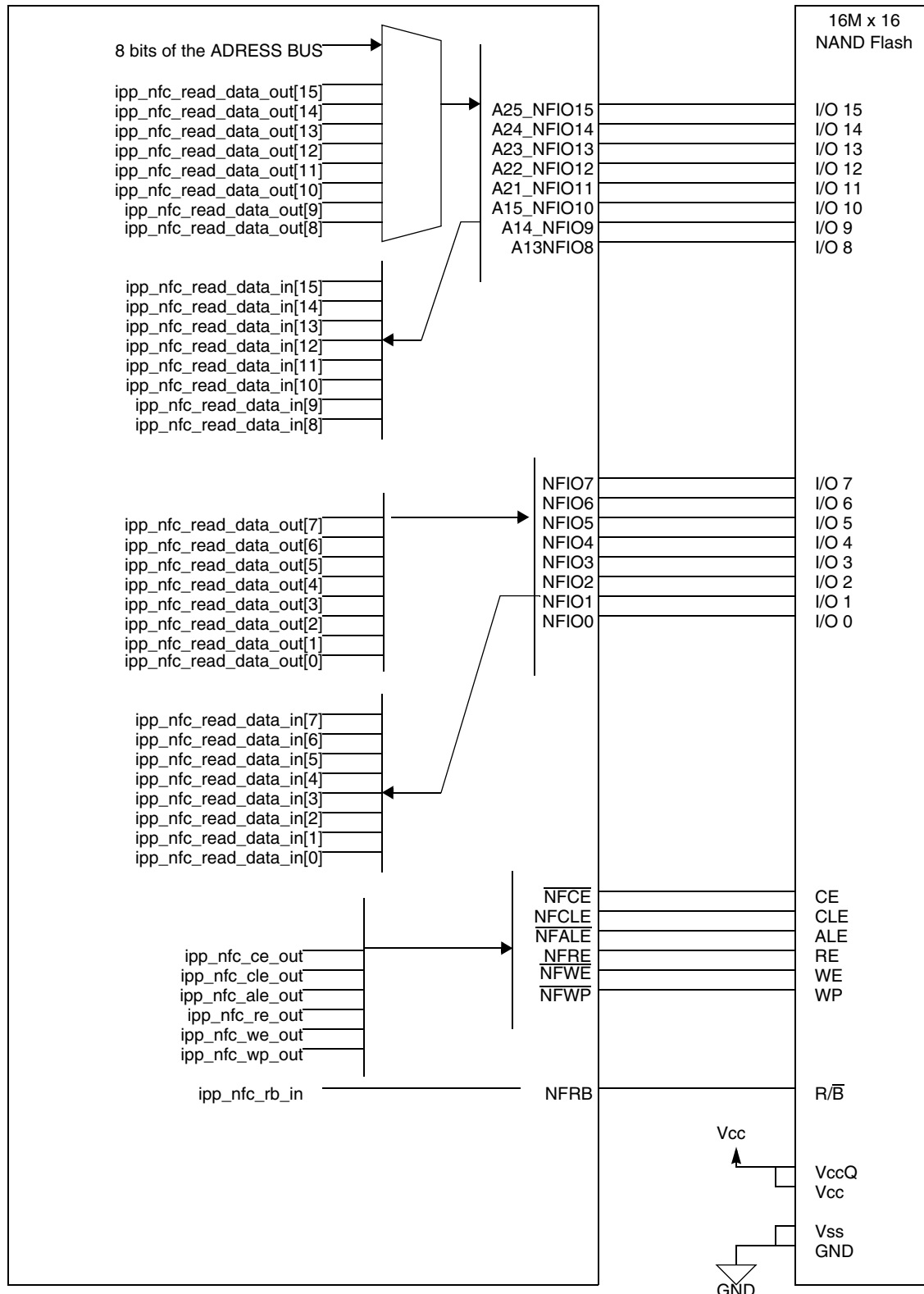


Figure 26-52. 256Mbit (16M x 16Bit) NAND Flash Connection Diagram

Chapter 27

Parallel Advanced Technology Attachment (PATA)

27.1 Introduction

Figure 27-1 shows a block diagram of the Parallel Advanced Technology Attachment (PATA) interface. This interface is primarily used to connect to IDE hard disc drives and ATAPI optical disc drives.

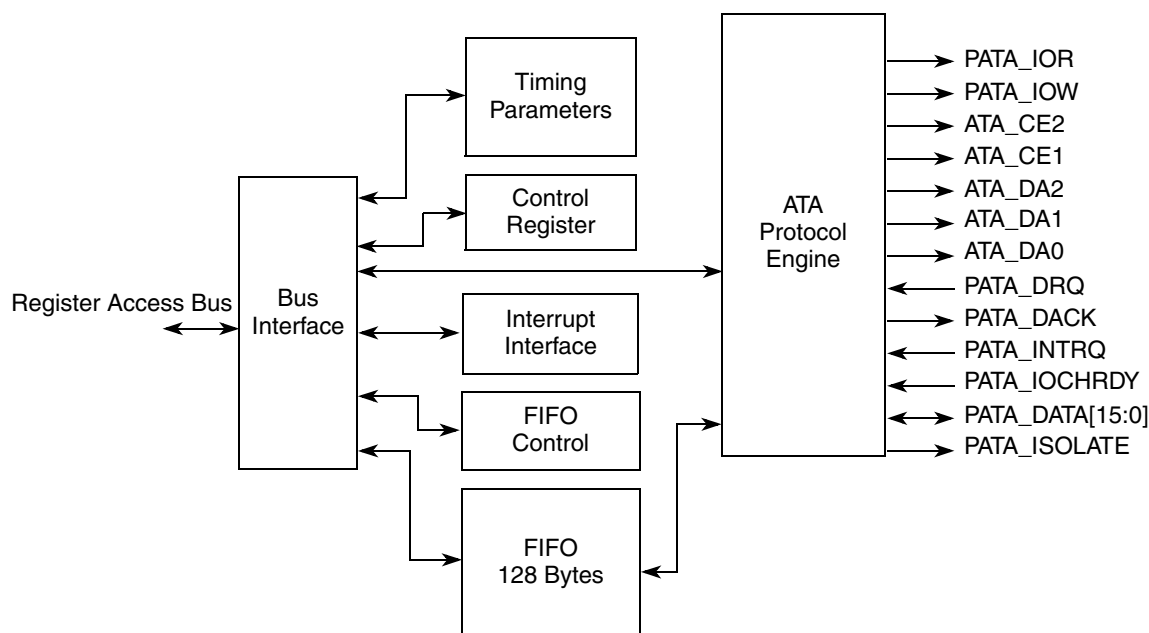


Figure 27-1. PATA Block Diagram

The PATA block communicates with the host processor and the host DMA unit via the register access bus. All internal PATA block registers are visible from register access bus.

Before making any access to the ATA bus, the host must program the PATA interface timing parameters on the ATA bus. The timing parameters control the timing on the ATA bus. Although some are implied, most timing parameters are programmable as a number of clock cycles (1 to 255).

After programming the timing parameters, two protocols can be activated at the same time on the ATA bus.

- The CPU core to the ATA bus can provide a parallel IO (PIO) mode access at any time. During PIO mode access, incoming register access bus cycle translates to an ATA bus cycle by the ATA protocol engine. The register access bus cycle stalls until completion of the ATA bus cycle on read or until putting the write data on the ATA bus on write. The PIO mode is a slow protocol (mainly intended to program the ATA disc drive), but also possible to transfer data to/from the disc drive. During PIO mode, internal FIFO is not active.
- The next protocol is the DMA mode access. DMA mode is started by the ATA interface after receiving a DMA request from the drive only if the ATA interface has been programmed to accept the DMA request. In DMA mode, multiword DMA or ultra DMA protocol is used on the ATA bus. After started, the data transfer is organized between the ATA bus and the FIFO. The data transfer pauses to prevent FIFO overflow/FIFO underflow, and it resumes when space exists again in the FIFO or when the FIFO has been refilled. During DMA transfer, no direct transfer between the ATA bus and IP bus occurs. Instead, the transfer happens between the ATA bus and the FIFO, and the FIFO informs the host DMA unit when it needs to be refilled or emptied.

When a PIO access occurs during a running DMA transfer, the DMA transfer is paused, the PIO access done, and then the DMA transfer resumes again.

27.1.1 Features

The ATA interface includes these features:

- Programmable timing on the ATA bus
- Compliant with ATA-6 specification
- Can be used with off-chip bus transceiver
- 128 byte FIFO part of interface
- FIFO receive alarm and FIFO transmit alarm to DMA unit

27.1.2 Modes of Operation

The interface offers two operation modes that can be active at the same time: PIO mode and DMA mode.

- PIO Mode:
 - An access to the ATA bus in PIO mode happens when the CPU core reads or writes an ATA PIO register. During a PIO transfer, the incoming register access bus cycle translates to an ATA PIO bus cycle by the ATA protocol engine. Because no data buffering occurs, the register bus cycle stalls until the ATA bus read data is available on read or the register bus data can be put on the ATA bus during write.
 - PIO accesses can happen to the bus at any time, even during a running ATA DMA transfer. In this case, the DMA transfer pauses, the PIO cycle completes, and the DMA transfer resumes.

- DMA Mode:
 - In DMA mode, data transfers between the ATA bus and the FIFO. Two different DMA protocols, ultra DMA mode and multiword DMA mode, are supported on the ATA bus. Selection occurs via a control register bit.
 - Writing a control bit enables DMA mode transfer and when the drive connects to the ATA bus, it pulls PATA_DRQ line high.
 - During an ATA bus DMA transfer, data transfers between the ATA bus and the FIFO. The transfer pauses to avoid FIFO overflow and FIFO underflow.
 - It is the task of the host DMA unit to read data or write data to the FIFO to maintain the transfer. For this purpose, the DMA write and read requests are sent to the host DMA unit. The DMA read request informs the host DMA unit of at least one packet of data waiting in the FIFO to be read by the host DMA. When this request is asserted, the host DMA should transfer one packet of data from the FIFO to the main memory. Typical packet size is 32 bytes (8 longwords), but other packet sizes can be managed too. The DMA write request informs the host DMA unit of space for at least one packet to be written by the host DMA. When this request is asserted, the host DMA should transfer one packet of data from main memory to the FIFO. Typical packet size is 32 bytes (8 longwords), but other packet sizes can be managed too.

27.2 External Signal Description

27.2.1 Overview

Table 27-2 provides detailed signal descriptions, but for a more detailed description of the ATA bus signal, refer to the ATA-6 specification.

Table 27-2. Detailed Signal Descriptions

Name	Description
PATA_IOR	PIO Mode – Read Strobe MDMA Mode – Read Strobe UDMA In Burst – HDMARDY UDMA Out Burst – Host Strobe
PATA_IOW	PIO Mode – Write Strobe MDMA – Write Strobe UDMA Burst – STOP Signal – asserted when the host wants to terminate a UDMA transfer
PATA_CE1	These signals are the address group of the ATA bus. ATA_CS0 and ATA_CS1 are the chip selects signals from the host that select the command block registers. ATA_DA2, ATA_DA1, and ATA_DA0 are the three address lines the host asserts to access a register or data port in the device.
PATA_CE0	
ATA_DA2	
ATA_DA1	
ATA_DA0	
PATA_DRQ	PATA DMA Request – This line is asserted to a logic 1 by an external device to request a multiword DMA (MDMA) or Ultra DMA (UDMA) transfer. 0 No DMA request 1 External PATA device is requesting the PATA bus

Table 27-2. Detailed Signal Descriptions

Name	Description
PATA_DACK	PATA Bus Host DMA Acknowledge. This signal is asserted to a logic 0 by the PATA HOST to grant the PATA Bus to an external device in response to a PATA DMA request. 0 Host is granting PATA bus to an external PATA device 1 Host has not granted PATA bus to an external PATA device.
PATA_INTRQ	PATA Bus Interrupt Request 0 Interrupt Asserted by external PATA Device 1 No interrupt Asserted by external PATA device
PATA_IOCHRDY	PATA bus IOCHRDY line. This pin has a different function for each following transfer mode: PIO Mode – IORD—active low wait during PIO cycles UDMA Mode (UDMA Out) – DDMARDY—active low device ready during ultra DMA out transfers UDMA Mode (UDMA In) – DSTROBE—device strobe during ultra DMA in transfers
PATA_DATA[15:0]	This is the ATA data bus.
PATA_ISOLATE	The PATA_ISOLATE pin controls an isolation buffer between the MPC5121e and the external PATA device. 0 Buffer drives data from the external PATA device to the MPC5121e. 1 Buffer drives data from the MPC5121e to the external PATA device.

27.2.2 Meeting Timing on the ATA Bus

Meeting timing requirements of the ATA bus requires special consideration because ATA devices usually connect to the PATA bus of the MPC5121e over a cable which introduces various types of delay and skew. In addition to this, there may be an isolation buffer between the external ATA device and the MPC5121e PATA bus. Because these delays depend on factors external to the MPC5121e, you must program the timing of the PATA bus so the timing specifications of the external ATA device are met. In general, ten basic timing diagrams must be considered.

They are:

- PIO read mode timing
- PIO write mode timing
- Multitword DMA read timing
- Multiword DMA write timing
- Ultradma start timing
- UltraDMA in host terminated transfer
- UltraDMA in device terminated transfer
- UltraDMA out transfer start timing
- UltraDMA out host terminated transfer
- UltraDMA out device terminated transfer.

To ensure timing specifications are met for the host and the target peripheral device, equations exist to determine the required PATA register values to properly adjust timing signals that take into account various timing skews and propagation delays between PATA Bus signals.

27.2.2.1 Timing Parameters

Table 27-3 shows various timing parameters affected by internal and external factors to the MPC5121e. One parameter, T, is the PATA bus clock period. This is the same as the LPC_CLK frequency when LPC_DIV of SCFR register in clock block is set to 3'b001. See section 6.4.1.4. Some parameters (ti_ds, tco, tskew2, etc.) are a function of the MPC5121e and the microcontroller top-level design controls them. Characteristics of the transceiver or isolation buffer between the MPC5121e and the external ATA device control other parameters. Also, characteristics of the ATA cable connecting the MPC5121e and the external ATA device controls other parameters.

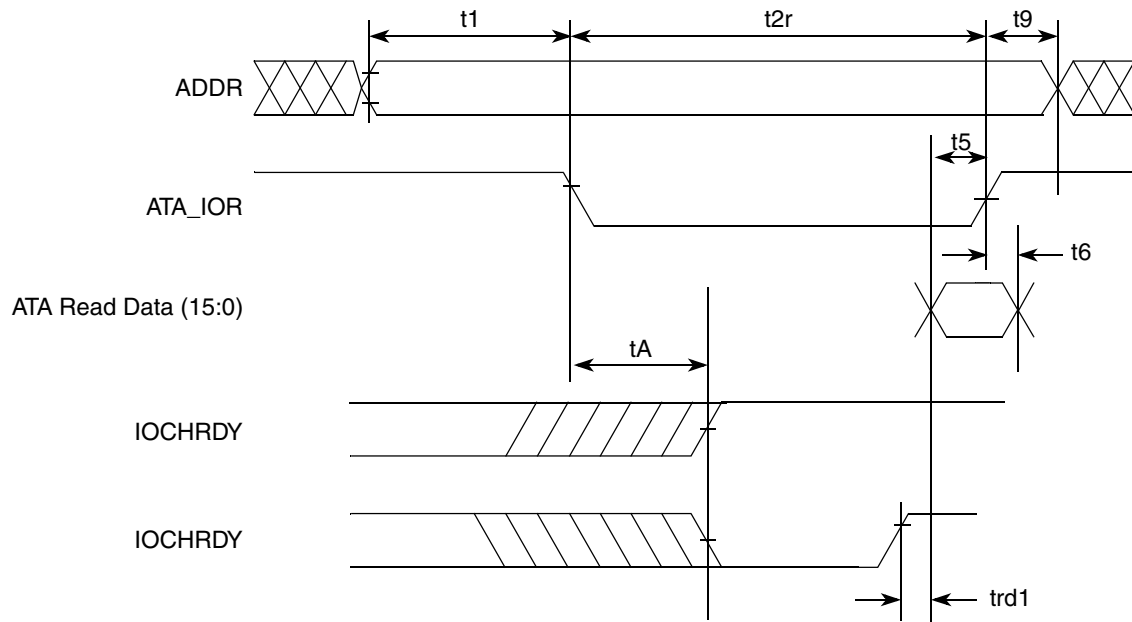
Most of the timing parameters controlling the various PATA bus signals are programmed in increments of the ATA bus clock period. A standard ATA bus clock frequency is 66 MHz, which has a period of 15 ns.

Table 27-3. Timing Parameters

Name	Meaning	Controlled by
T	PATA Bus clock period (Same as LocalPlus Bus Clock)	Clock generator
ti_ds	Internal set-up time PATA_DATA to PATA_IOCHRDY negative edge (UDMA-in only)	Top level design
ti_dh	Internal hold time PATA_IOCHRDY negative edge to PATA_DATA invalid (UDMA-in only)	Top level design
tco	Propagation delay between positive edge of ata bus clock to ATA_CE1, ATA_CE2, ATA_DA2, ATA_DA1, ATA_DA0, PATA_IOR, PATA_IOW, PATA_DACK, PATA_DATA, PATA_ISOLATE	Top level design
tsu	Set-up time from PATA_DATA valid to positive edge of ATA Bus Clock	Top level design
tsui	Set-up time from PATA_IOCHRDY to negative edge of ATA Bus clock	Top level design
thi	Hold time from PATA_IOCHRDY to negative edge of ATA Bus clock	Top level design
tskew1	Max difference in propagation delay between positive edge of ATA Bus Clock to ATA_CE1, ATA_CE0, ATA_DA2, ATA_DA1, ATA_DA0, PATA_IOR, PATA_IOW, PATA_DACK, PATA_DATA (write), PATA_ISOLATE	Top level design
tskew2	Max difference in buffer propagation delay for ATA_CE1 ATA_CE2, ATA_DA2, ATA_DA1, ATA_DA0, PATA_DIOR, PATA_DIOW, PATA_DMACK, PATA_DATA (write), PATA_ISOLATE	Transceiver
tskew3	Max difference in buffer propagation delay for PATA_IOCHRDY, PATA_DATA (read)	Transceiver
tbuf	Max buffer propagation delay	Transceiver
tcable1	Cable propagation delay for PATA_DATA	Cable
tcable2	Cable propagation delay for control signals PATA_IOR, PATA_IOW, PATA_IOCHRDY, PATA_DACK	Cable
tskew4	Max difference in cable propagation delay between PATA_IOCHRDY and PATA_DATA (read)	Cable
tskew5	Max difference in cable propagation delay between (PATA_IOR, PATA_IOW, PATA_DACK) and ATA_CE1, ATA_CE2, ATA_DA2, ATA_DA1, ATA_DA0, PATA_DATA(write)	Cable
tskew6	Max difference in cable propagation delay without accounting for ground bounce	Cable

27.2.2.2 PIO Mode Timing

Figure 27-2 shows a timing diagram for the PIO read mode.


Figure 27-2. PIO Read Mode Timing

To fulfill read mode timing, observe the different timing parameters in [Table 27-4](#).

Table 27-4. Timing Parameters PIO Read

ATA Parameter	Figure 27-2 Parameter	Value	How to meet
t1	t1	$t1(\min) = \text{time_1} * T - (\text{tskew1} + \text{tskew2} + \text{tskew5})$	Calculate and programming time_1, see 27.3.2.1/27-17
t2	t2r	$t2(\min) = \text{time_2r} * T - (\text{tskew1} + \text{tskew2} + \text{tskew5})$	Calculate and programming time_2r, see 27.3.2.2/27-18
t9	t9	$t9(\min) = \text{time_9} * T - (\text{tskew1} + \text{tskew2} + \text{tskew6})$	Calculate and programming time_9, see 27.3.2.3/27-19
t5	t5	$t5(\min) = \text{tco} + \text{tsu} + \text{tbuf} + \text{tbuf} + \text{tcable1} + \text{tcable2}$	If not met, increase time_2r
t6	t6	0	—
tA	tA	$tA(\min) = (1.5 + \text{time_ax}) * T - (\text{tco} + \text{tsui} + \text{tcable2} + \text{tcable2} + 2 * \text{tbuf})$	Calculate and programming time_ax, see 27.3.2.2/27-18
trd	trd1	$\text{trd1}(\max) = (-\text{trd}) + (\text{tskew3} + \text{tskew4})$ $\text{trd1}(\min) = (\text{time_pio_rdx} - 0.5) * T - (\text{tsu} + \text{thi})$ $(\text{time_pio_rdx} - 0.5) * T > \text{tsu} + \text{thi} + \text{tskew3} + \text{tskew4}$	Calculate and programming time_pio_rdx, see 27.3.2.2/27-18
t0	—	$t0(\min) = (\text{time_1} + \text{time_2} + \text{time_9}) * T$	time_1, time_2r, time_9

[Figure 27-3](#) shows timing waveforms are somewhat different in PIO write mode.

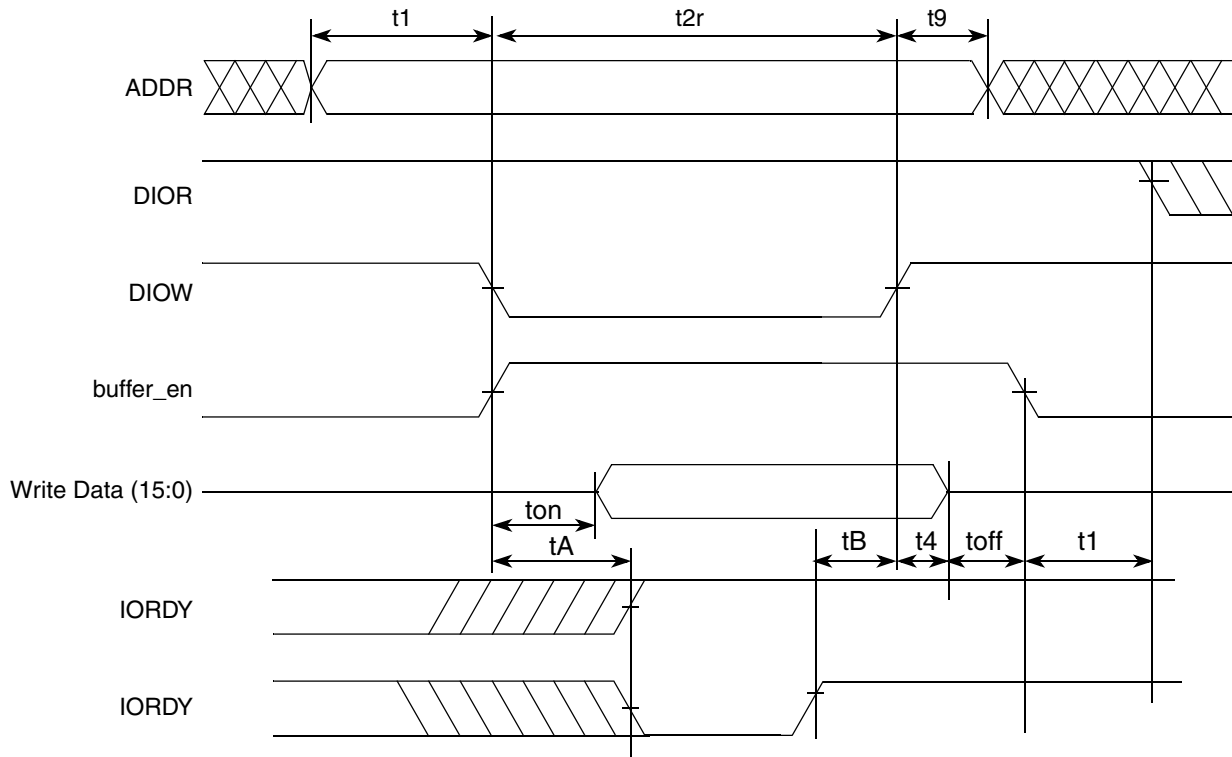


Figure 27-3. PIO Write Mode Timing

Table 27-5 shows several parameters need to be observed to fulfill this timing.

Table 27-5. Timing Parameters PIO Write

ATA Parameter	Figure 27-3 Parameter	Value	How to meet
t1	t1	$t1(\text{min}) = \text{time_1} * T - (\text{tskew1} + \text{tskew2} + \text{tskew5})$	time_1, see 27.3.2.1/27-17
t2	t2r	$t2(\text{min}) = \text{time_2w} * T - (\text{tskew1} + \text{tskew2} + \text{tskew5})$	Calculate and programming time_2w, see 27.3.2.1/27-17
t9	t9	$t9(\text{min}) = \text{time_9} * T - (\text{tskew1} + \text{tskew2} + \text{tskew6})$	time_9, see 27.3.2.3/27-19
t3	—	$t3(\text{min}) = (\text{time_2w} - \text{time_on}) * T - (\text{tskew1} + \text{tskew2} + \text{tskew5})$	If not met, increase time_2w
t4	t4	$t4(\text{min}) = \text{time_4} * T - \text{tskew1}$	Calculate and programming time_4, see 27.3.2.2/27-18
tA	tA	$tA = (1.5 + \text{time_ax}) * T - (\text{tco} + \text{tsui} + \text{tcable2} + \text{tcable2} + 2 * \text{tbuf})$	Calculate and programming time_ax, see 27.3.2.2/27-18
t0	—	$t0(\text{min}) = (\text{time_1} + \text{time_2} + \text{time_9}) * T$	time_1, time_2r, time_9
—	—	Avoid bus contention when switching buffer on by making ton long enough	—
—	—	Avoid bus contention when switching buffer off by making toff long enough	—

27.2.2.3 Timing in Multiword DMA Mode

Figure 27-4 and Figure 27-5 give timing in multiword DMA mode.

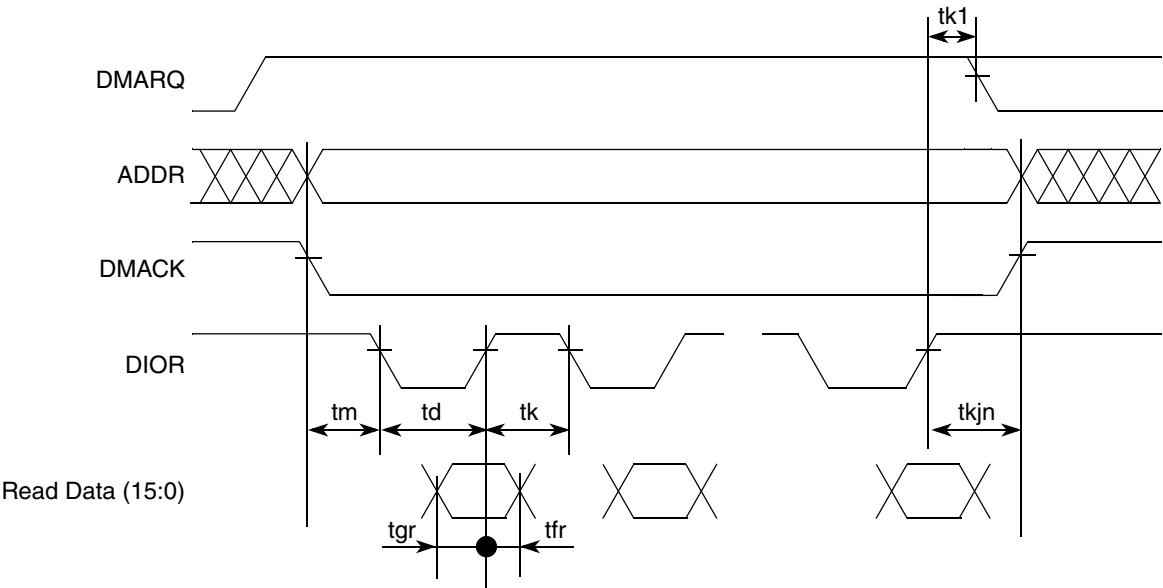


Figure 27-4. MDMA Read Timing

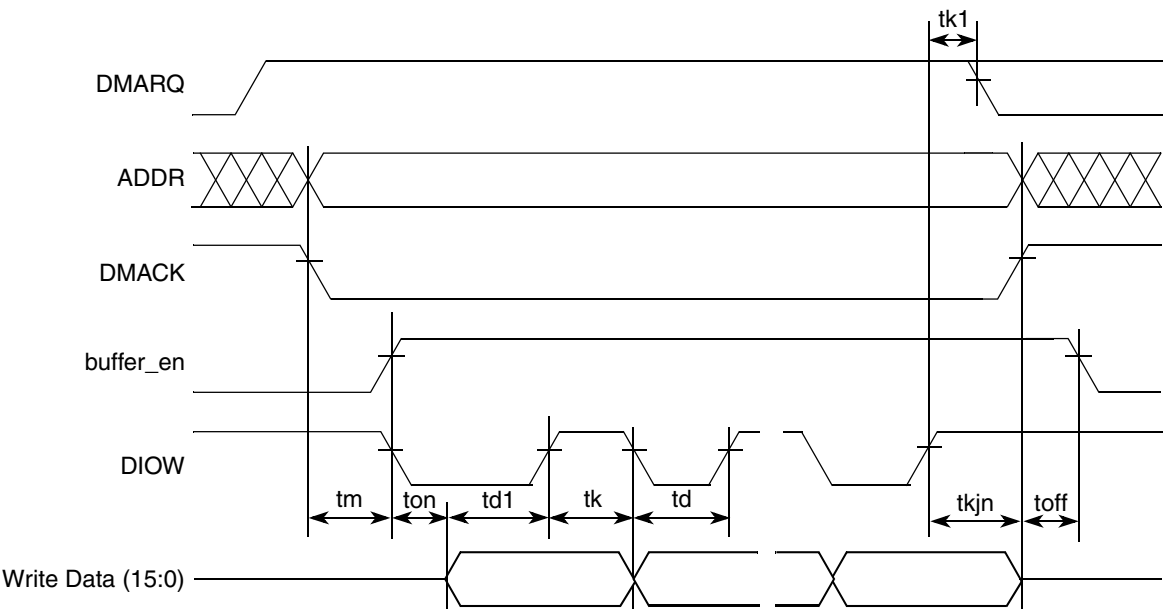


Figure 27-5. MDMA Write Timing

To meet this timing, a number of timing parameters must be controlled as shown in Table 27-6.

Table 27-6. Timing Parameters MDMA Read and Write

ATA Parameter	Figure 27-4 Figure 27-5 Parameter	Value	How to meet
tm, ti	tm	$tm(min) = ti(min) = time_m * T - (tskew1 + tskew2 + tskew5)$	Calculate and programming time_m, see 27.3.2.3/27-19
td	td, td1	$td1(min) = td(min) = time_d * T - (tskew1 + tskew2 + tskew6)$	Calculate and programming time_d, see 27.3.2.3/27-19
tk	tk	$tk(min) = time_k * T - (tskew1 + tskew2 + tskew6)$	Calculate and programming time_k, see 27.3.2.4/27-19
t0	—	$t0(min) = (time_d + time_k) * T$	time_d, time_k
tg(read)	tgr	$tgr(min-read) = tco + tsu + tbuf + tbuf + tcable1 + tcable2$ $tgr(min-drive) = td - te(drive)$	time_d, see 27.3.2.3/27-19
tf(read)	tfr	$tfr(min-drive) = 0$	—
tg(write)	—	$tg(min-write) = time_d * T - (tskew1 + tskew2 + tskew5)$	time_d
tf(write)	—	$tf(min-write) = time_k * T - (tskew1 + tskew2 + tskew6)$	time_k
tL	—	$tL(max) = (time_d + time_k - 2) * T - (tsu + tco + 2 * tbuf + 2 * tcable2)$	time_d, time_k
tn, tj	tkjn	$tn = tj = tkjn = (max(time_k, time_jn) * T - (tskew1 + tskew2 + tskew6))$	Calculate and programming time_jn, see 27.3.2.3/27-19
—	ton toff	$ton = time_on * T - tskew1$ $toff = time_off * T - tskew1$	—

27.2.2.4 UDMA In Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA in transfer are given:

- [Figure 27-6](#) gives timing for UDMA in transfer start
- [Figure 27-7](#) gives timing for host terminating UDMA in transfer
- [Figure 27-8](#) gives timing for device terminating UDMA in transfer.

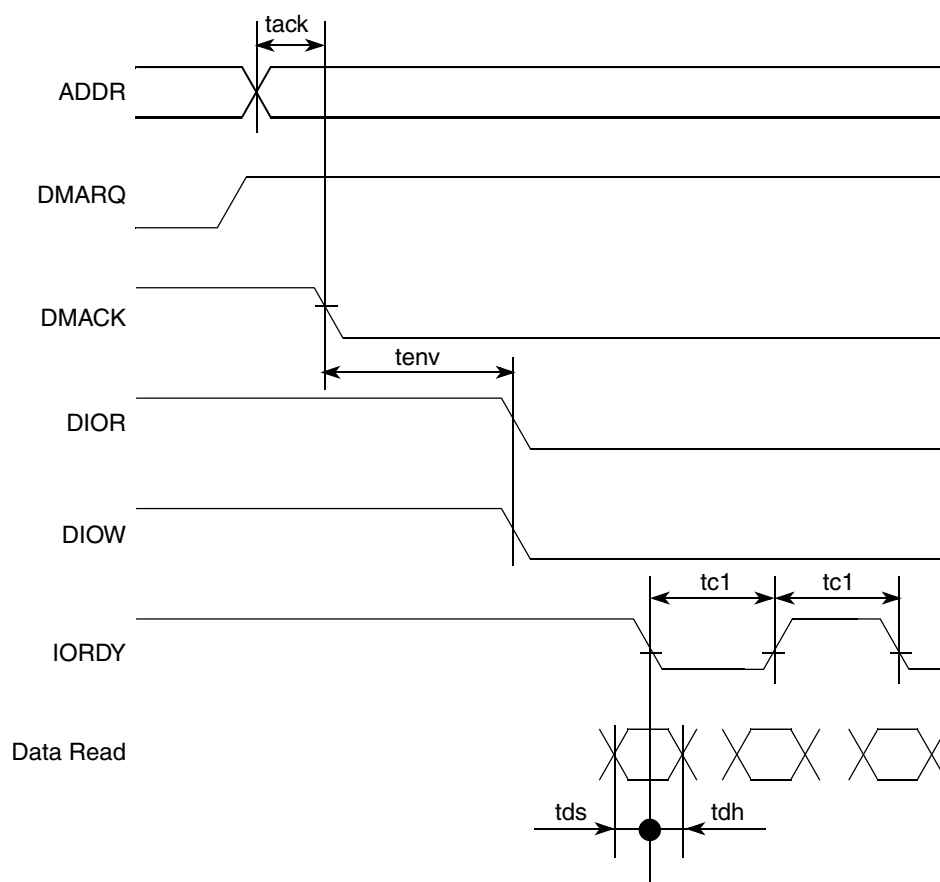


Figure 27-6. UDMA In Transfer Start Timing Diagram

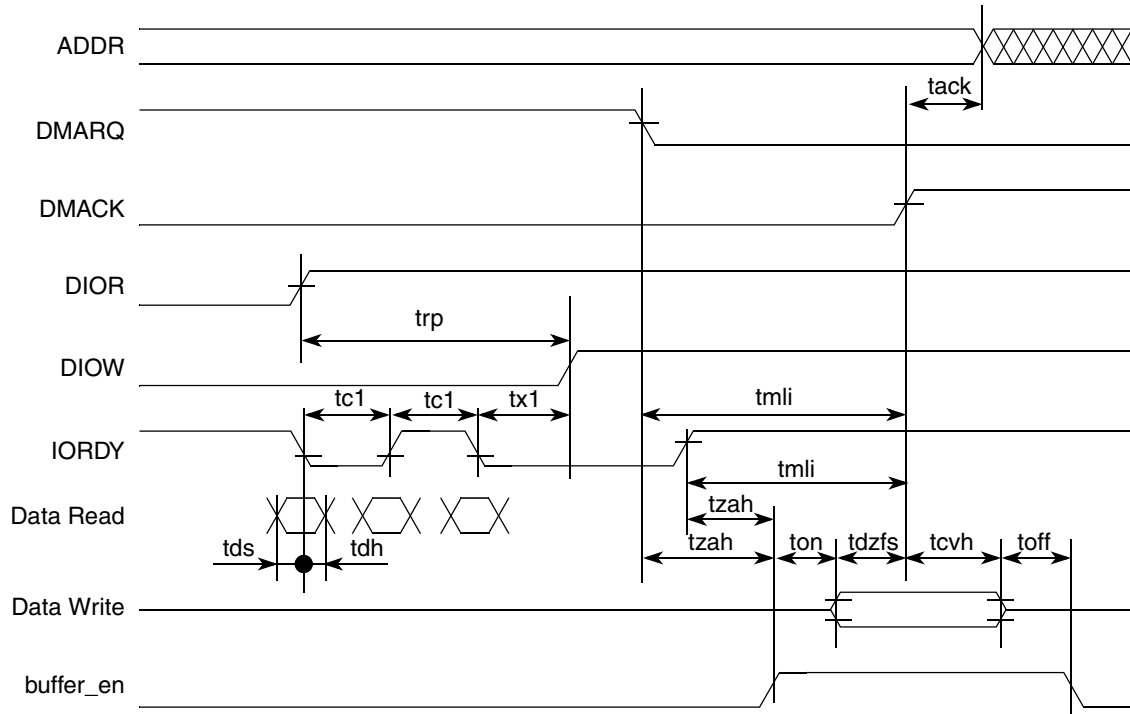


Figure 27-7. UDMA In Host Terminates Transfer

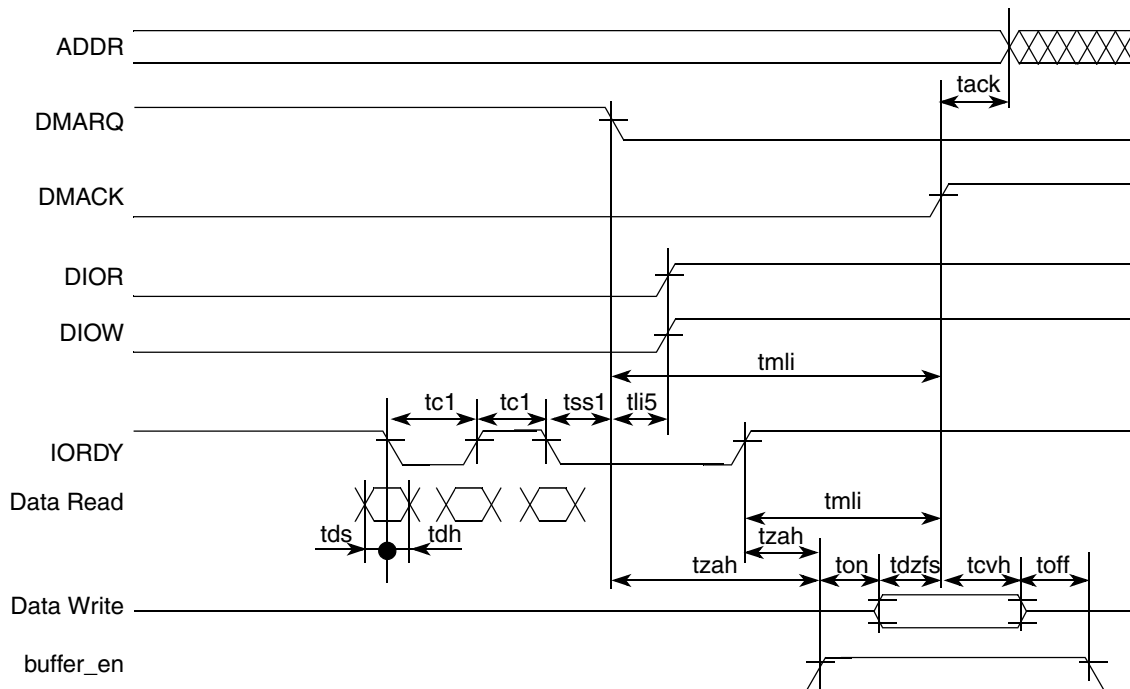


Figure 27-8. UDMA In Device Terminates Transfer

Table 27-7 explains timing parameters.

Table 27-7. Timing Parameters UDMA in Burst

ATA Parameter	Figure 27-6 Figure 27-7 Figure 27-8 Parameter	Value	How to Meet
tack	tack	$tack(min) = (time_ack * T) - (tskew1 + tskew2)$	Calculate and programming time_ack, see 27.3.2.4/27-19
tenv	tenv	$tenv(min) = (time_env * T) - (tskew1 + tskew2)$ $tenv(max) = (time_env * T) + (tskew1 + tskew2)$	Calculate and programming time_env, see 27.3.2.4/27-19
tds	tds1	$tds - (tskew3) - ti_ds > 0$	tskew3, ti_ds, ti_dh should be low enough
tdh	tdh1	$tdh - (tskew3) - ti_dh > 0$	
tcyc	tc1	$(tcyc - tskew) > T$	Bus clock period T big enough
trp	trp	$trp(min) = time_rp * T - (tskew1 + tskew2 + tskew6)$	calculate and programming time_rp, see 27.3.2.4/27-19
	tx1 ¹	$(time_rp * T) - (tco + tsu + 3T + 2 * tbuf + 2 * tcable2) > trfs (drive)$	Calculate and programming time_rp, see 27.3.2.4/27-19
tmli	tmli1	$tmli1(min) = (time_mlix + 0.4) * T$	Calculate and programming time_mlix, see 27.3.2.5/27-20
tzah	tzah	$tzah(min) = (time_zah + 0.4) * T$	Calculate and programming time_zah, see 27.3.2.5/27-20
tdzfs	tdzfs	$tdzfs = (time_dzfs * T) - (tskew1 + tskew2)$	Calculate and programming time_dzfs, see 27.3.2.5/27-20
tcvh	tcvh	$tcvh = (time_cvh * T) - (tskew1 + tskew2)$	Calculate and programming time_cvh, see 27.3.2.6/27-20
—	ton toff ²	$ton = time_on * T - tskew1$ $toff = time_off * T - tskew1$	—

¹ A special timing requirement in the ATA host requires the internal DIOW to go only high three clocks after the last active edge on the DSTROBE signal. The equation given on this line captures this constraint.

² Make ton and toff big enough to avoid bus contention.

27.2.2.5 UDMA Out Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA out transfer are given:

- [Figure 27-9](#) gives timing for UDMA out transfer start
- [Figure 27-10](#) gives timing for host terminating UDMA out transfer
- [Figure 27-11](#) gives timing for device terminating UDMA out transfer.

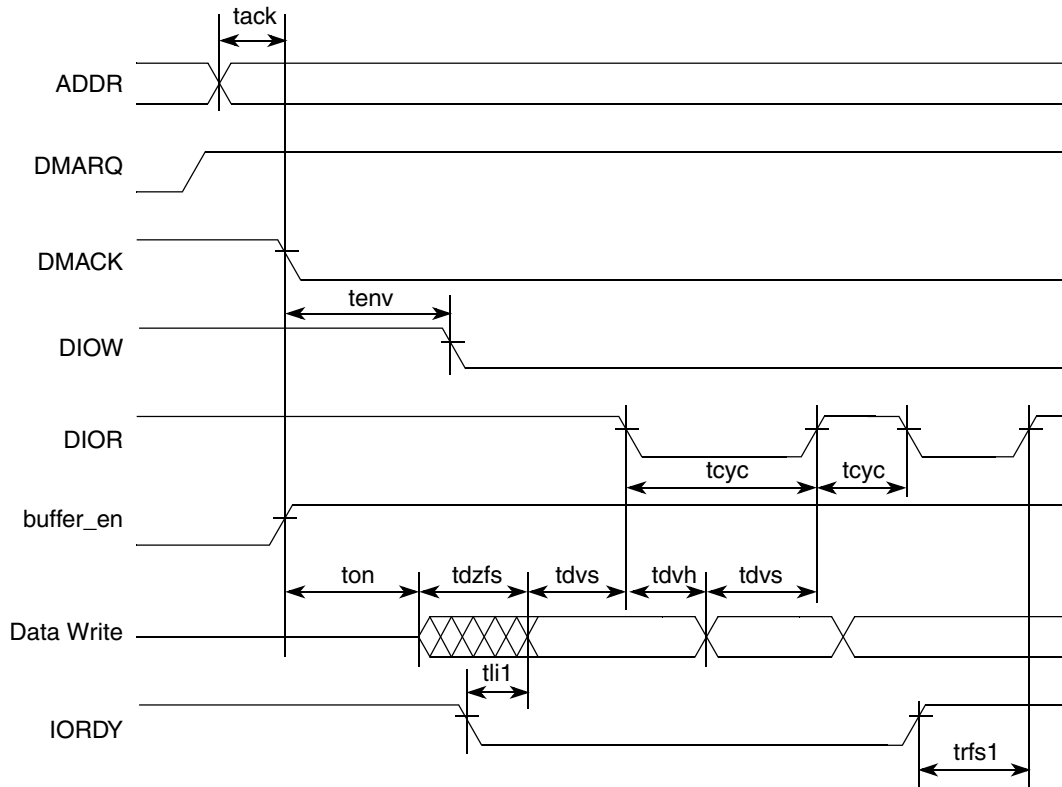


Figure 27-9. UDMA Out Transfer Start Timing Diagram

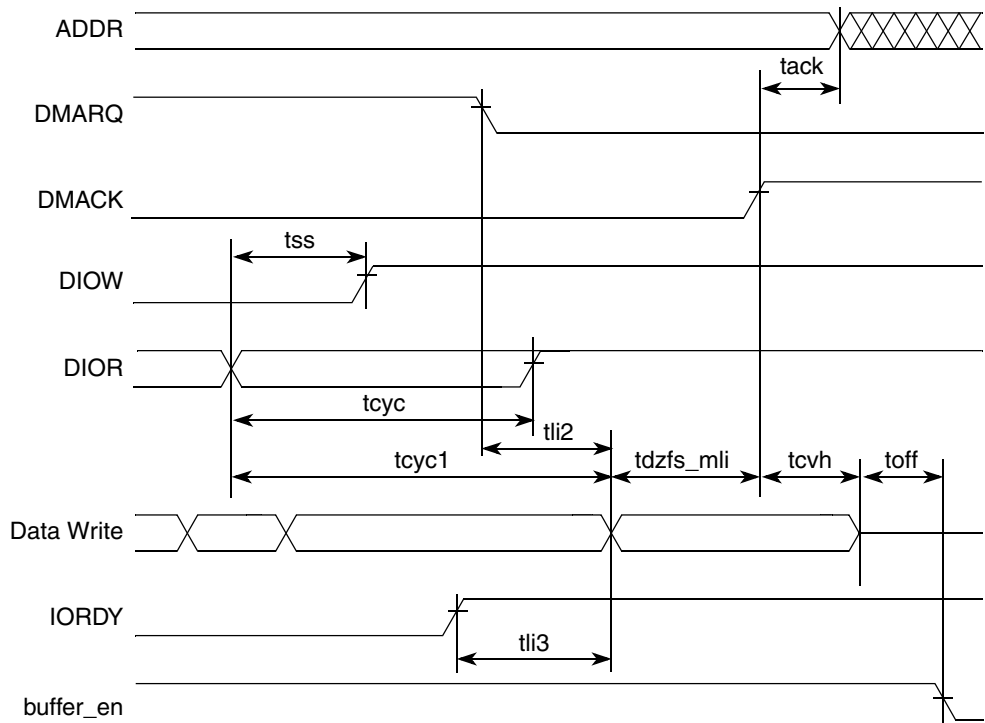


Figure 27-10. UDMA Out Host Terminates Transfer

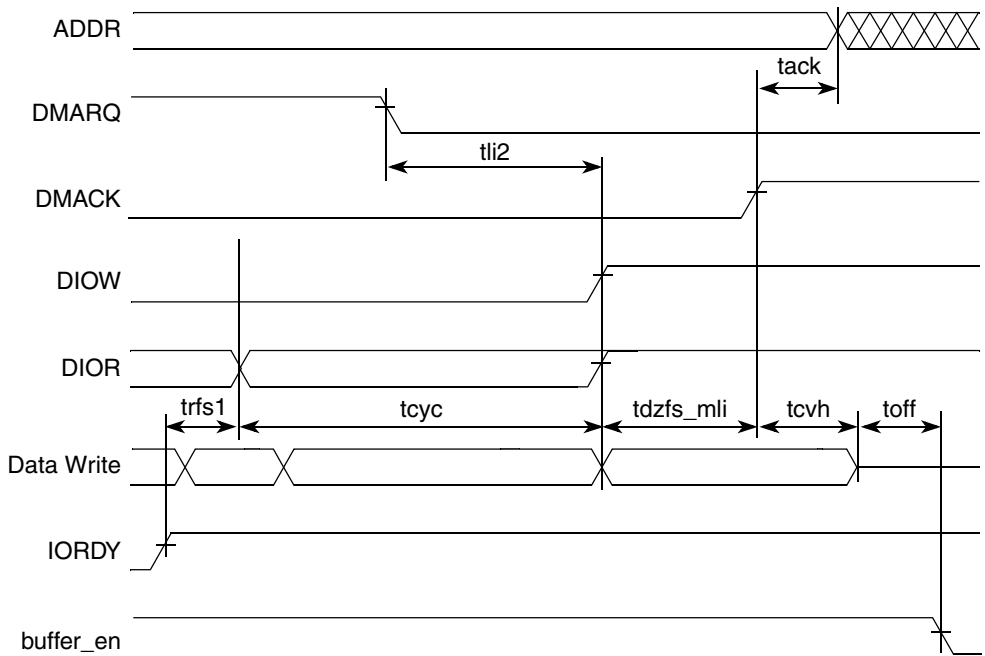


Figure 27-11. UDMA Out Device Terminates Transfer

Table 27-8 explains timing parameters.

Table 27-8. Timing Parameters UDMA Out Burst

ATA Parameter	Figure 27-9 Figure 27-10 Figure 27-11 Parameter	Value	How to meet
tack	tack	$tack(min) = (time_ack * T) - (tskew1 + tskew2)$	Calculate and Programming Time_ack, See 27.3.2.4/27-19
tenv	tenv	$tenv(min) = (time_env * T) - (tskew1 + tskew2)$ $tenv(max) = (time_env * T) + (tskew1 + tskew2)$	Calculate and Programming Time_env, See 27.3.2.4/27-19
tdvs	tdvs	$tdvs = (time_dvs * T) - (tskew1 + tskew2)$	Calculate and Programming Time_dvs, See 27.3.2.6/27-20
tdvh	tdvh	$tdvs = (time_dvh * T) - (tskew1 + tskew2)$	Calculate and Programming Time_dvh, See 27.3.2.5/27-20
tcyc	tcyc	$tcyc = time_cyc * T - (tskew1 + tskew2)$	Calculate and Programming Time_cyc, See 27.3.2.6/27-20
t2cyc		$t2cyc = time_cyc * 2 * T$	Calculate and Programming Time_cyc, See 27.3.2.6/27-20
trfs1	trfs	$trfs = 1.6 * T + tsui + tco + tbuf + tbuf$	—
-	tdzfs	$tdzfs = time_dzfs * T - (tskew1)$	Calculate and Programming Time_dzfs, See 27.3.2.5/27-20
tss	tss	$tss = time_ss * T - (tskew1 + tskew2)$	Calculate and Programming Time_ss, See 27.3.2.6/27-20
tmli	tdzfs_mli	$tdzfs_mli = \max(time_dzfs, time_mli) * T - (tskew1 + tskew2)$	—
tli	tli1	$tli1 > 0$	—
tli	tli2	$tli2 > 0$	—
tli	tli3	$tli3 > 0$	—
tcvh	tcvh	$tcvh = (time_cvh * T) - (tskew1 + tskew2)$	Calculate and Programming Time_cvh, See 27.3.2.6/27-20
—	ton toff	$ton = time_on * T - tskew1$ $toff = time_off * T - tskew1$	—

27.3 Memory Map and Register Definition

27.3.1 Memory Map

Table 27-9 is the memory map for the ATA module.

Table 27-9. Module Memory Map

Address	Register	Access	Section/Page
(ata_base + 0x00)	Time register1: PIO and transceiver timing parameter.	R/W	27.3.2.1/27-17
(ata_base + 0x04)	Time register2: PIO timing parameter.	R/W	27.3.2.2/27-18
(ata_base + 0x08)	Time register3: PIO and MDMA timing parameter.	R/W	27.3.2.3/27-19
(ata_base + 0x0C)	Time register4: MDMA and UDMA timing parameter.	R/W	27.3.2.4/27-19
(ata_base + 0x10)	Time register5: UDMA timing parameter.	R/W	27.3.2.5/27-20
(ata_base + 0x14)	Time register6: UDMA timing parameter.	R/W	27.3.2.6/27-20
(ata_base + 0x18)	FIFO_DATA_32: 32-bit data port to/from FIFO	R/W	27.3.2.7/27-21
(ata_base + 0x1C)	FIFO_DATA_16: 16-bit data port to/from FIFO	R/W	27.3.2.8/27-21
(ata_base + 0x20)	FIFO_FILL: FIFO filling in halfwords	R	27.3.2.9/27-22
(ata_base + 0x24)	ATA_CONTROL: ATA interface control register	R/W	27.3.2.10/27-23
(ata_base + 0x28)	INTERRUPT_PENDING: Interrupt pending register	R	27.3.2.11/27-25
(ata_base + 0x2C)	INTERRUPT_ENABLE: Interrupt enable register	R/W	27.3.2.12/27-25
(ata_base + 0x30)	INTERRUPT_CLEAR: Interrupt clear register	W	27.3.2.13/27-26
(ata_base + 0x34)	FIFO_ALARM: FIFO alarm threshold	R/W	27.3.2.14/27-27
(ata_base + 0xA0)	DRIVE_DATA: drive data register	R/W	27.3.2.15/27-28
(ata_base + 0xA4)	DRIVE_FEATURES: drive features register	R/W	27.3.2.15/27-28
(ata_base + 0xA8)	DRIVE_SECTOR_COUNT: drive sector count register	R/W	27.3.2.15/27-28
(ata_base + 0xAC)	DRIVE_SECTOR_NUM: drive sector number register	R/W	27.3.2.15/27-28
(ata_base + 0xB0)	DRIVE_CYL_LOW: drive cylinder low register	R/W	27.3.2.15/27-28
(ata_base + 0xB4)	DRIVE_CYL_HIGH: drive cylinder high register	R/W	27.3.2.15/27-28
(ata_base + 0xB8)	DRIVE_DEV_HEAD: drive device head register	R/W	27.3.2.15/27-28
(ata_base + 0xBC)	DRIVE_COMMAND: drive command register	W	27.3.2.15/27-28
(ata_base + 0xBC)	DRIVE_STATUS: drive status register	R	27.3.2.15/27-28
(ata_base + 0xD8)	DRIVE_ALT_STATUS: drive alternate status register	R	27.3.2.15/27-28
(ata_base + 0xD8)	DRIVE_CONTROL: drive control register	W	27.3.2.15/27-28

27.3.2 Register Descriptions

The following sections show details of the PATA registers. The PATA clock must be turned on in the system clock control register 1 before writing into any PATA register.

27.3.2.1 Timing Register 1

Figure 27-12 contains part of the timing parameters on the ATA bus. See Table 27-10 for a description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1. Calculate the value of every field according to bus clock period and the timing determined in ATA specification, then program it via IP bus. It is the same for all timing registers.

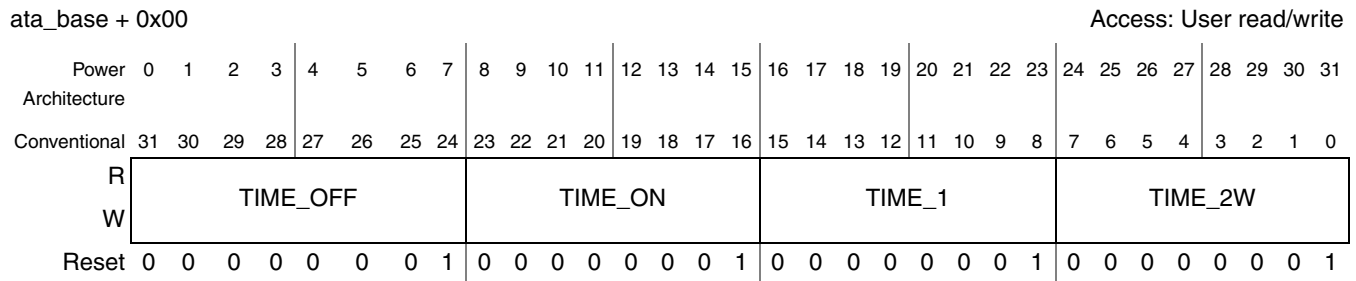


Figure 27-12. Timing Register 1

Table 27-10. Timing Register 1 Field Descriptions

Field	Description
TIME_2W	PIO timing parameter. Controls t ₂ during write cycles. The t _{2w} is the minimum command active time. The time PATA_DIOW is 0 here.
TIME_1	PIO timing parameter. Controls t ₁ . The t ₁ is the cycle time from address valid to PATA_DIOR/PATA_DIOW setup.
TIME_ON	Transceiver timing parameter. Controls t _{On} .
TIME_OFF	Transceiver timing parameter. Controls t _{Off} .

27.3.2.2 Timing Register 2

Figure 27-12 contains part of the timing parameters on the ATA bus. See Table 27-11 for a description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1.

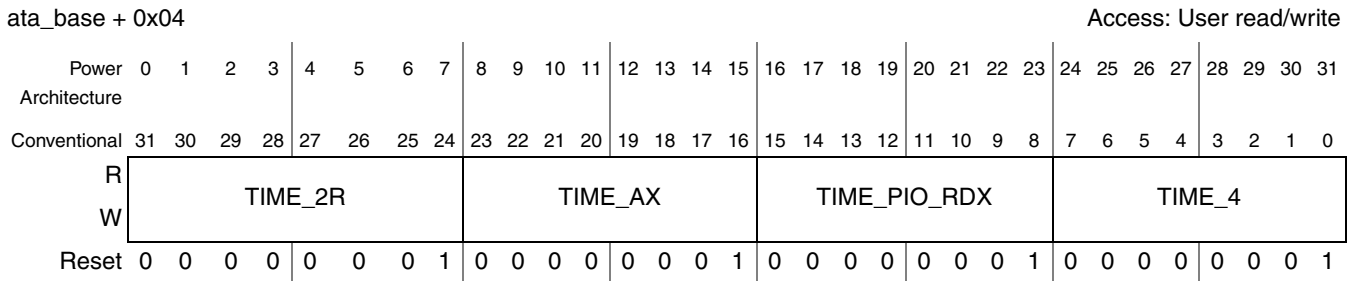


Figure 27-13. Timing Register 2

Table 27-11. Timing Register 2 Field Descriptions

Field	Description
TIME_4	PIO timing parameter. Controls t4. The t4 is write data hold time.
TIME_PIO_RDX	PIO timing parameter. Controls trd. The trd is the minimum time from read data valid to PATA_IOCHRDY active.
TIME_AX	PIO timing parameter. Controls tA. The tA is PATA_IOCHRDY setup time.
TIME_2R	PIO timing parameter. Controls t2 during read cycles. The t2r is the minimum command active time. Here is the time PATA_DIOR equals to 0.

27.3.2.3 Timing Register 3

Figure 27-14 contains part of timing parameters on the ATA bus. See Table 27-12 for a description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1.

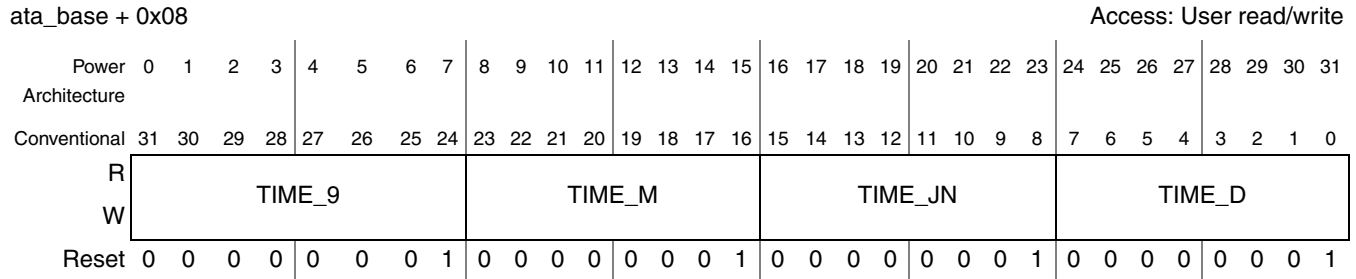


Figure 27-14. Timing Register 3

Table 27-12. Timing Register 3 Field Descriptions

Field	Description
TIME_D	MDMA timing parameter. Controls td.
TIME_JN	MDMA timing parameter. Controls tn and tj.
TIME_M	MDMA timing parameter. Controls tm.
TIME_9	PIO timing parameter. Controls t9.

27.3.2.4 Timing Register 4

Figure 27-15 contains part of timing parameters on the ATA bus. See Table 27-13 for a description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1.

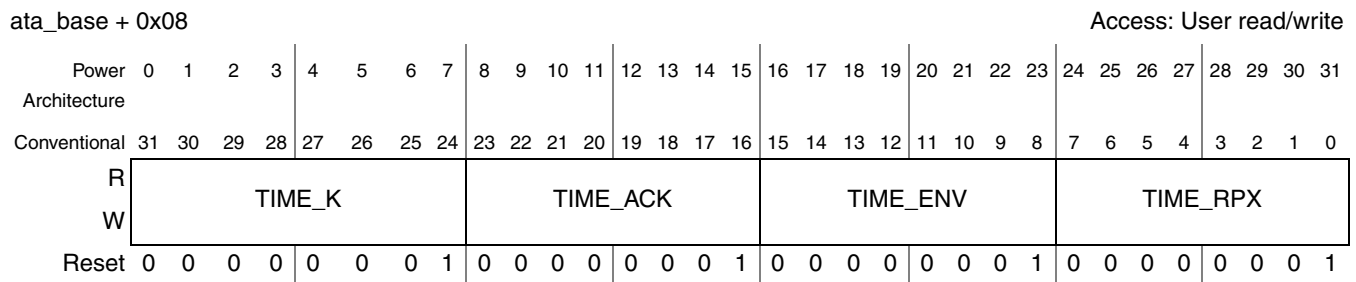


Figure 27-15. Timing Register 4

Table 27-13. Timing Register 4 Field Descriptions

Field	Description
TIME_RPX	UDMA timing parameter. Controls trp.
TIME_ENV	UDMA timing parameter. Controls tn and tenv.
TIME_ACK	UDMA timing parameter. Controls tack.
TIME_K	MDMA timing parameter. Controls tk.

27.3.2.5 Timing Register 5

Figure 27-16 contains part of timing parameters on the ATA bus. See Table 27-14 for a description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1.

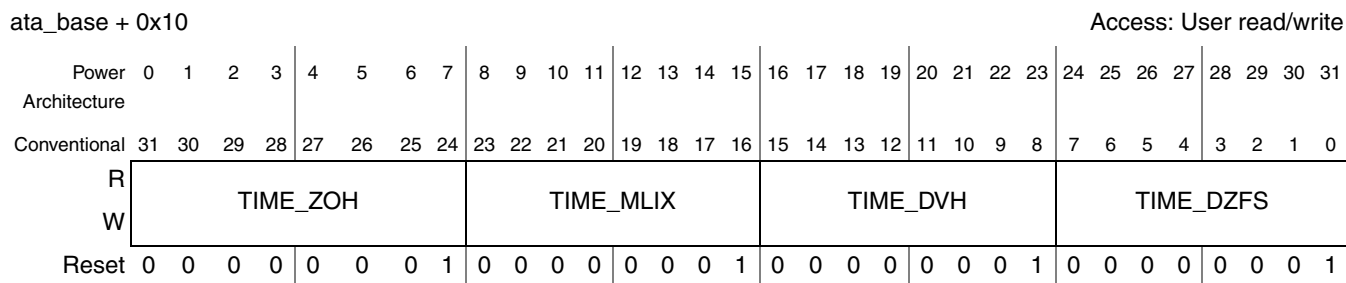


Figure 27-16. Timing Register 5

Table 27-14. Timing Register 5 Field Descriptions

Field	Description
TIME_DZFS	UDMA timing parameter. Controls tdzfs.
TIME_DVH	UDMA timing parameter. Controls tdvh.
TIME_MLIX	UDMA timing parameter. Controls tml.
TIME_ZAH	UDMA timing parameter. Controls tzah.

27.3.2.6 Timing Register 6

Figure 27-17 contains part of timing parameters on the ATA bus. See Table 27-15 for description of the bits fields. Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset values are always 1.

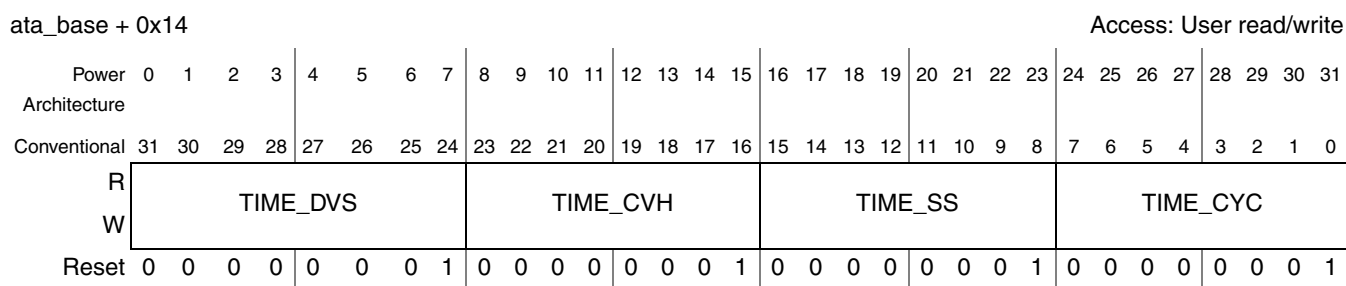


Figure 27-17. Timing Register 6

Table 27-15. Timing Register 6 Field Descriptions

Field	Description
TIME_CYC	UDMA timing parameter. Controls tcyc and t2cyc.
TIME_SS	UDMA timing parameter. Controls tss.
TIME_CVH	UDMA timing parameter. Controls tcvh.
TIME_DVS	UDMA timing parameter. Controls tdvs.

27.3.2.7 FIFO Data 32 Register

The FIFO_DATA_32 register shown in Figure 27-18 is used by the 32-wide data port to write to/read from the FIFO. See Table 27-16 for a description of the bit fields.

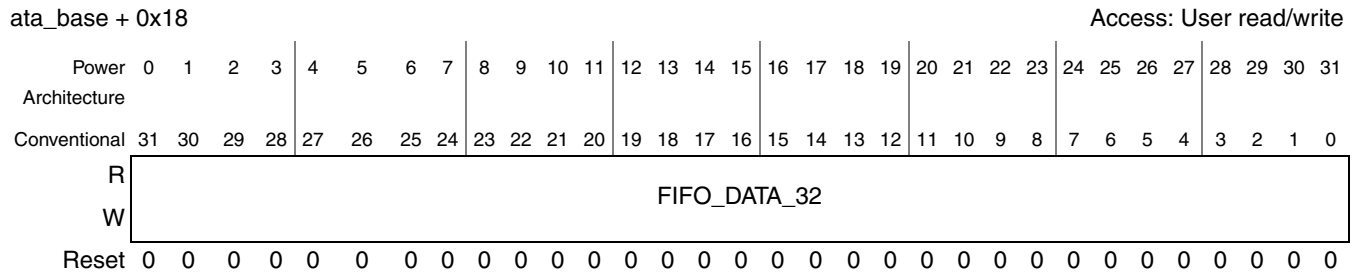


Figure 27-18. FIFO Data 32 Register

Table 27-16. FIFO Data 32 Field Descriptions

Field	Description
FIFO_DATA_32	The FIFO_REGISTER_32 is used to read or write data to the internal FIFO. Access it as a 32-bit register. Any write to the register puts the 4 bytes written into the FIFO. Any read access fetches 4 bytes from the FIFO.

27.3.2.8 FIFO Data 16 Register

The FIFO_DATA_16 register shown in Figure 27-19 is used by the 16-wide data port to write to/read from the FIFO. See Table 27-17 for a description of the bit fields.

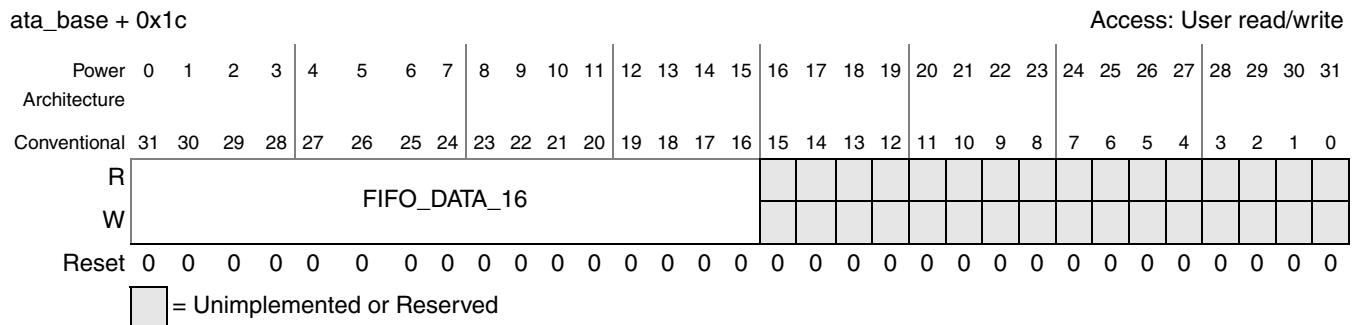


Figure 27-19. FIFO Data 6 Register

Table 27-17. FIFO Data 16 Field Descriptions

Field	Description
FIFO_DATA_16	The FIFO_DATA_16 register reads or writes data to the internal FIFO. Access it as a 16-bit register. Any write to the register puts the 2 bytes written into the FIFO. Any read access fetches read 2 bytes from the FIFO.

27.3.2.9 FIFO Fill Register

The FIFO fill register shown in Figure 27-20 is a read-only register. See Table 27-18 for a description of the bit fields.

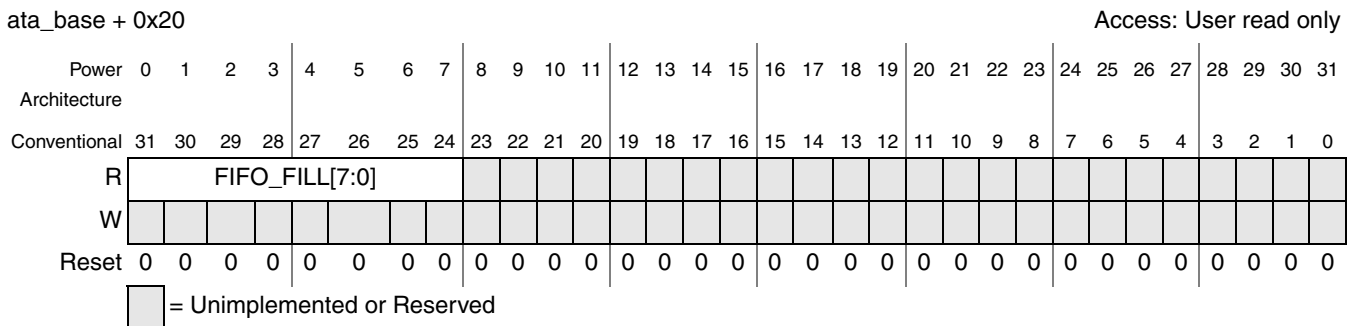


Figure 27-20. FIFO_FILL Register

Table 27-18. FIFO FILL Field Descriptions

Field	Description
FIFO_FILL[7:0]	FIFO_FILL is a read-only register. Any read to it returns the current number of 16-bit half-words present in the FIFO.

27.3.2.10 ATA_CONTROL Register

Figure 27-21 shows the ATA control register. See Table 27-19 for a description of the bit fields.

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	FIFO_RST_B	ATA_RST_B	FIFO_TX_EN	FIFO_RCV_EN	DMA_PENDING	DMA_ULTRA_SELECTED	DMA_WRITE	IORDY_EN								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 27-21. ATA Control Register
(Register repeats for reference.)

Table 27-19. ATA Control Field Descriptions

Field	Description
FIFO_RST_B	FIFO reset. This field controls if the internal FIFO is in reset or enabled. 1 FIFO normal operation 0 FIFO reset
ATA_RST_B	ATA reset. This field provides a way that software can reset the internal ata protocol engine. 1 ATA_RESET_B=1, internal protocol engine normal operation. 0 ATA_RESET_B=0, internal protocol engine reset Note: ATA_RST_B must be set before accessing a PATA drive register. Otherwise, an error exception occurs.
FIFO_TX_EN	FIFO transmit enable. This field controls if the FIFO makes transmit data requests to the DMA. If enabled, the FIFO requests the DMA to refill it when FIFO filling drops below the alarm level. 1 FIFO refill by DMA enabled 0 FIFO refill by DMA disabled
FIFO_RCV_EN	FIFO receive enable. This field controls if the FIFO makes receive data requests to the DMA. If enabled, the FIFO requests the DMA to empty it when FIFO filling becomes greater or equal to the alarm level. 1 FIFO empty by DMA enabled 0 FIFO empty by DMA disabled
DMA_PENDING	DMA pending. This field controls if the ATA interface responds to a DMA request originating in the drive. If this bit is asserted, the ATA interface starts a multiword DMA or ultra DMA burst when the drive asserts ATA_DMAARQ. 1 ATA interface starts multiword DMA or ultra DMA burst when drive asserts dmarq 0 ATA interface does not start DMA burst

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	FIFO_RS_T_B	ATA_RST_B	FIFO_TX_EN	FIFO_RC_V_EN	DMA_PE_NDING	DMA_ULTRA_SELECTED	DMA_WRITE	IORDY_EN								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 27-21. ATA Control Register
(Register repeats for reference.)

Table 27-19. ATA Control Field Descriptions (continued)

Field	Description
DMA_ULTRA_SELECTED	DMA ultra selected. This bit indicates if a DMA burst started and whether the UDMA or MDMA protocol is used. 1 ultra DMA protocol is used 0 multiword DMA protocol is used
DMA_WRITE	DMA write. This bit indicates the data direction on any DMA burst started. 1 DMA out burst, ATA interface writes to drive 0 DMA in burst, ATA interface reads from drive
IORDY_EN	IO ready enable. This bit indicates if the ATA_IORDY handshake is used during PIO mode 1 ATA_IORDY handshake is used 0 ATA_IORDY is disregarded.

27.3.2.11 Interrupt Pending Register

Figure 27-22 shows the interrupt pending register. See Table 27-20 for a description of the bit fields.

Offset 0x28 Access: User read only

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		FIFO_UN ERFLOW	FIFO_OV ERFLOW	CONTRO LLER_ID LE	ATA_IRTR Q2											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

□ = Unimplemented or Reserved

Figure 27-22. Interrupt Pending Register

27.3.2.12 Interrupt Enable Register

Figure 27-23 shows the interrupt enable register. See Table 27-20 for a description of the bit fields.

Offset 0x2C Access: User read/write

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		FIFO_UN DERFLO W	FIFO_OV ERFLOW	CONTRO LLER_ID LE	ATA_IRTR Q2											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 27-23. Interrupt Enable Register

27.3.2.13 Interrupt Clear Register

Figure 27-24 shows the interrupt clear register. See Table 27-20 for a description of the bit fields.

Offset 0x30 Access: User write only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W		FIFO_UNDERFLOW	FIFO_OVERFLOW													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 27-24. Interrupt Clear Register

These three registers control interrupts coming from the ATA and going to the CPU: interrupt pending, interrupt enable, and interrupt clear.

The interrupt-pending register and the interrupt-enable register controls the interrupt interface from the ATA module.

- Bits 27, 28, 29 and 30 of the interrupt registers control this interrupt. The interrupt is asserted if one of the four bits is set in the interrupt pending register while the same bit is set in the interrupt_enable register. This interrupt goes to the CPU.

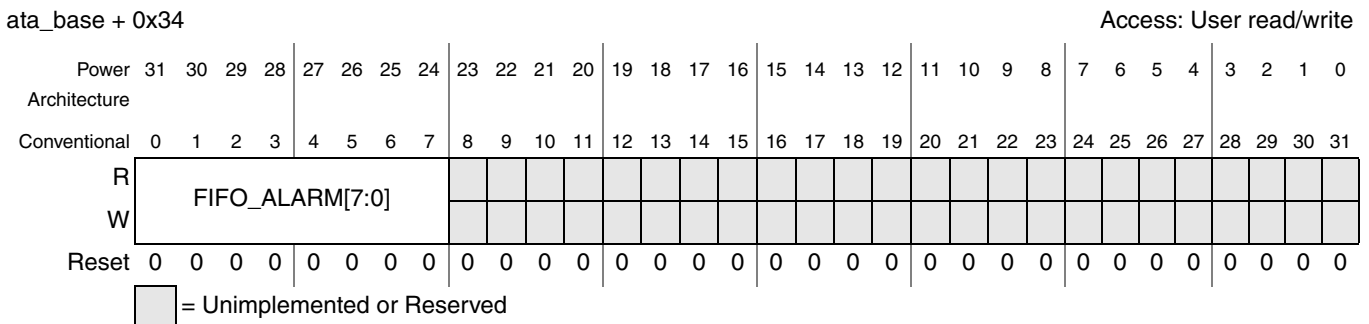
These registers have mostly the same bits. If a bit is set in the interrupt pending register, its interrupt is pending and it produces an interrupt if the same bit is set in the interrupt enable register. Some bits in the interrupt pending register are sticky bits. Writing a 1 to the corresponding bit in the interrupt clear bit, resets them.

Table 27-20. Interrupt Register Field Descriptions

Field	Description
FIFO_UNDERFLOW	FIFO underflow. This bit reports FIFO underflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO underflow condition. It is cleared by writing a 1 to this bit in the interrupt clear register. When the bit is set in the interrupt pending register and the same bit is set in the interrupt enable register, an interrupt signals to the CPU.
FIFO_OVERFLOW	FIFO overflow. This bit reports FIFO overflow. Sticky bit. It is set in the interrupt pending register when there is a FIFO overflow condition. It is cleared by writing a 1 to this bit in the interrupt clear register. When the bit is set in the interrupt pending register and the same bit is set in the interrupt enable register, an interrupt signals to the CPU.
CONTROLLER_IDLE	Controller idle. This bit reports controller idle. It is set when the ATA protocol engine is idle, there is no activity on the ATA bus. It is cleared when there is activity on the ATA bus. When the bit is set in the interrupt pending register and the same bit is set in the interrupt enable register, an interrupt signals to the CPU. The interrupt clear register has no influence on this bit.
ATA_INTRQ2	ATA interrupt request. This bit reflects the value of the ATA_INTRQ interrupt input. It is set in the interrupt pending register when the drive interrupt is pending and cleared otherwise. When the bit is set in the interrupt pending register and the same bit is set in the interrupt enable register, an interrupt signals that the CPU the drive is requesting attention. The interrupt clear register has no influence on this bit.

27.3.2.14 FIFO Alarm Register

The FIFO_alarm register in [Figure 27-25](#) is the threshold to generate an alarm to the DMA interface. See [Table 27-21](#) for a description of the bit fields.

**Figure 27-25. FIFO Alarm Register****Table 27-21. FIFO Alarm Field Descriptions**

Field	Description
FIFO_ALARM[7:0]	This register contains the threshold to generate DMA write and DMA read requests to the DMA interface. If (FIFO_TX_EN == 1 && FIFO_FILL < FIFO_ALARM): DMA write request is made to DMA to refill FIFO. If (FIFO_RCV_EN == 1 && FIFO_FILL >= FIFO_ALARM): DMA read request is made to DMA to empty FIFO. Note: The threshold value is expressed in 16-bit half-words.

27.3.2.15 Registers Present in the Drive Connected to the ATA Bus

Some registers are addressable, but are not present in the PATA module. [Table 27-22](#) gives a list of these registers. If a read or write access is made to one of these registers, the read or write maps to a PIO read or write cycle on the ATA bus. The corresponding register in the device attached to the ATA bus is accessed. No description of operation of these registers is given here. Consult the ATA specification for all details here.

Table 27-22. Registers Present in the Drive Connected to the ATA Bus

Address	Name	Description	Access
(ata_base + 0xA0)	DRIVE_DATA	Drive data register	32-bit RW
(ata_base + 0xA4)	DRIVE_FEATURES	Drive features register	32-bit RW
(ata_base + 0xA8)	DRIVE_SECTOR_COUNT	Drive sector count register	32-bit RW
(ata_base + 0xAC)	DRIVE_SECTOR_NUM	Drive sector number register	32-bit RW
(ata_base + 0xB0)	DRIVE_CYL_LOW	Drive cylinder low register	32-bit RW
(ata_base + 0xB4)	DRIVE_CYL_HIGH	Drive cylinder high register	32-bit RW
(ata_base + 0xB8)	DRIVE_DEV_HEAD	Drive device head register	32-bit RW
(ata_base + 0xBC)	DRIVE_COMMAND	Drive command register	32-bit W
(ata_base + 0xBC)	DRIVE_STATUS	Drive status register	32-bit R
(ata_base + 0xD8)	DRIVE_ALT_STATUS	Drive alternate status register	32-bit R
(ata_base + 0xD8)	DRIVE_CONTROL	Drive control register	32-bit W

27.4 Functional Description

The ATA interface provides two ways to communicate with the ATA peripherals connected to the ATA bus:

- PIO mode read/write operation to the ATA bus
- DMA transfers with the ATA bus

The following subsections describe and detail the operation of the peripheral.

27.4.1 Reset

Besides hardware reset, software can write bit 6 ATA_RST_B of register ATA control to reset PATA block. When ATA_RST_B is cleared to 0, the ATA protocol engine is reset. When this bit is set to 1, the reset is released.

27.4.2 Programming ATA Bus Timing and IORDY_EN

The timing the ATA interface operates with on the ATA bus is programmable. The programming uses the timing registers at ata_base+0x0 to ata_base+0x17. [Section 27.2.2, “Meeting Timing on the ATA Bus”](#) details how these registers affect the timing parameters on the ATA bus. Reprogramming of these registers is allowed at any time when the ATA bus is idle. Before reprogramming, make sure:

- DMA_PENDING in ATA control register is cleared

- CONTROLLER_IDLE in interrupt pending register is set

Accomplish these two conditions by writing DMA_PENDING to 0, waiting until CONTROLLER_IDLE is set, and reprogramming the timing parameters. If DMA_PENDING was 1 before the reprogramming started, it should be set again after new timing is in effect to allow the drive to finish the current DMA transfer.

Reprogram the bus timing during an ongoing DMA transfer when necessary because the operating system wants to change the bus clock period.

It is necessary to wait for CONTROLLER_IDLE because a PIO read or write to the ATA bus terminates after the bus cycle when the CPU has been terminated. If the wait for CONTROLLER_IDLE is not done, the new timing values may affect a bus cycle that continues to run and cause error.

The IORDY_EN bit in register ATA control influences whether the ATA interface listens to the IORDY signal coming from the drive. To reprogram it, apply the same rules as applied to the timing registers:

- Only allowed when DMA_PENDING is cleared
- While CONTROLLER_IDLE is set

27.4.3 Access to ATA Bus in PIO Mode

Access to the ATA bus in PIO mode is possible after:

- ATA_RST_B in ATA control register is set
- Timing parameters have been programmed

To access the drive in PIO mode, read or write to the correct drive register. The bus cycle is translated to an ATA cycle, and the drive is accessed. When drive registers are accessed while the ATA bus is in reset, the read or write is discarded, not done.

27.4.4 Using DMA Mode to Receive Data from the ATA Bus

Apart from PIO mode, the ATA interface also supports MDMA and UDMA mode to transfer data. Use DMA mode to receive data from the drive (DMA in transfer). In DMA receive mode, the protocol engine transfers data from the drive to the FIFO using multiword DMA or ultra DMA protocol. The transfer pauses when:

- The FIFO is full
- The drive deasserts its dma request signal PATA_DMARQ.
- DMA_PENDING in the ATA control register clear.

When the cause of the transfer pausing is removed, the transfer restarts. The drive signals to the host at the end of transfer by asserting the PATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive also indicates if transfer ends.

The host system DMA manages the transfer of data from the FIFO into the memory. When the FIFO filling is above the alarm threshold, DMA should read one packet of data from the FIFO and store this in main memory. In doing so, DMA prevents the FIFO from getting full and keeps the transfer from drive to FIFO running.

This list describes the steps to set up a DMA data transfer from device to host:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty or by resetting it.
3. Every time a DMA read request is asserted, the DMA should read <packetsize> long ints(4 bytes) from the FIFO and store them to main memory. (packetsize means the number of doublewords transferred per DMA request, typical value is 8)
4. Write $2 * \text{<packetsize>}$ to FIFO alarm register. In this way, FIFO requests attention to DMA when at least one packet is ready for transfer.
5. To make the ATA ready for a DMA transfer from device to host:
 - Make sure the FIFO is out of reset by setting FIFO_RST_B to 1 in the ata control register.
 - Program FIFO_RCV_EN=1 in ATA control register. This enables DMA to empty the FIFO.
 - Program DMA_PENDING =1, DMA_WRITE=0, ULTRA_MODE_SELECTED=0/1 in ATA_CONTROL register. ULTRA_MODE_SELECTED should be 1 if you want to transfer data using UDMA mode. It should be 0 if you want to transfer data using MDMA mode.
6. Now, the host side of the DMA is ready. Send commands to the drive in PIO mode that cause it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. Please consult the ATA specification to know how to communicate with the drive.
7. When the drive requests DMA transfer by pulling PATA_DRQ high, the ATA interface acknowledges with PATA_DACK, and the transfer starts. Data transfers automatically to the FIFO and to the host memory from there.
8. During the transfer, the host can monitor for transfer status by reading some device ATA registers. These reads cause the running DMA to pause, complete the read, and the DMA resumes. The host can also wait until the drive asserts PATA_INTRQ. This also indicates end of transfer.
9. On end of transfer, the host or host DMA should wait until the host sees CONTROLLER_IDLE is set, and next read the remaining halfwords from the FIFO, and transfer these to memory.

NOTE

On end of transfer, there may be less than <packetsize> remaining bytes.
They cannot be transmitted automatically by the DMA.

27.4.5 Using DMA Mode to Transmit Data to the ATA Bus

In DMA mode, the host can transmit data to the drive (DMA out transfer). In DMA transfer mode, the protocol engine transfers data from the FIFO to the drive using multword DMA or ultra DMA protocol. The transfer pauses when one of following occurs:

- The FIFO is empty
- The drive deasserts its dma request signal PATA_DMARQ.
- The bit DMA_PENDING in the ATA contol register is cleared.

When the cause of the transfer pausing is removed, the transfer restarts. The drive signals to the host the end of the transfer by asserting the PATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive also indicates if transfer ends.

The host DMA system manages the transfer of data from the memory to the FIFO. When the FIFO filling is below the alarm threshold, the DMA should read one packet of data from the main memory and store this in the FIFO. In doing so, the DMA prevents the FIFO from getting empty and keeps the transfer from FIFO to drive running.

This list describes the steps to set up a DMA data transfer from device to host:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty, or by resetting it.
3. Every time a DMA read request is asserted the DMA should read <packetsize> long ints(4 bytes) from the main memory and write them to the FIFO. (packetsize means the number of doublewords transferred per DMA request, typical value is 8). Program the DMA so it does not transfer more than <sectorsize> longwords in total.
4. Write $\text{FIFO_SIZE} - 2 * \text{<packetsize>}$ to FIFO_ALARM register. In this way, FIFO requests attention to DMA when room exists for at least one extra packet. FIFO_SIZE should be given in halfwords. It is 64 in MPC5151E.
5. To make the ATA ready for a DMA transfer from host to device:
 - Make sure the FIFO is out of reset by setting bit FIFO_rst_b to 1 in the ata_control register.
 - Program FIFO_TX_EN=1 in ata_control register. This enables DMA to fill the FIFO.
 - Program DMA_PENDING =1, DMA_WRITE=1, ULTRA_MODE_SELECTED=0/1 in ata_control register. The ULTRA_MODE_SELECTED should be 1 if you want to transfer data using UDMA mode. It should be 0 if you want to transfer data using MDMA mode.
6. Now, the host side of the DMA is ready. Send commands to the drive in PIO mode that cause it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. Please consult the ATA specification to know how to communicate with the drive.
7. When the drive now requests DMA transfer by pulling PATA_DMARQ high, the ATA interface acknowledges with PATA_DACK and the transfer starts. Data is transferred automatically from the FIFO and from host memory to FIFO.
8. During the transfer, the host can monitor for status of transfer by reading some device ATA registers. These reads cause the running DMA to pause, the read to complete, and then DMA resumes. The host can also wait until the drive asserts PATA_INTRQ. This also indicates end of transfer.

Chapter 28

PCI Controller (PCI)

28.1 Introduction

The PCI controller connects the processor and memory system to the I/O components via the PCI system bus. This interface acts as initiator (master) and target (slave) device. The PCI controller uses a 32-bit multiplexed, address/data bus. The interface provides address and data parity with error checking and reporting. The interface provides for three physical address spaces – 32-bit address memory, 32-bit address I/O, and PCI configuration space. It supports up to three external masters.

Figure 28-1 is a block diagram of the PCI.

28.1.1 Features

The PCI includes the following features:

- PCI specification revision 2.3 compliant
- 32-bit PCI interface support on primary PCI port
- On-chip arbitration supports three external PCI bus masters (in addition to the PCI itself)
- Arbiter supports two-level priority request/grant signal pairs
- Supports accesses to all PCI address spaces
- Supports PCI-to-local and local-to-PCI streaming
- Memory prefetching of PCI read accesses and support for delayed read transactions
- Supports posting of processor to PCI and PCI to memory writes
- Supports selectable snooping for inbound transactions
- PCI host bridge capabilities
- Address translation units for address mapping between the PCI bus and the internal bus.

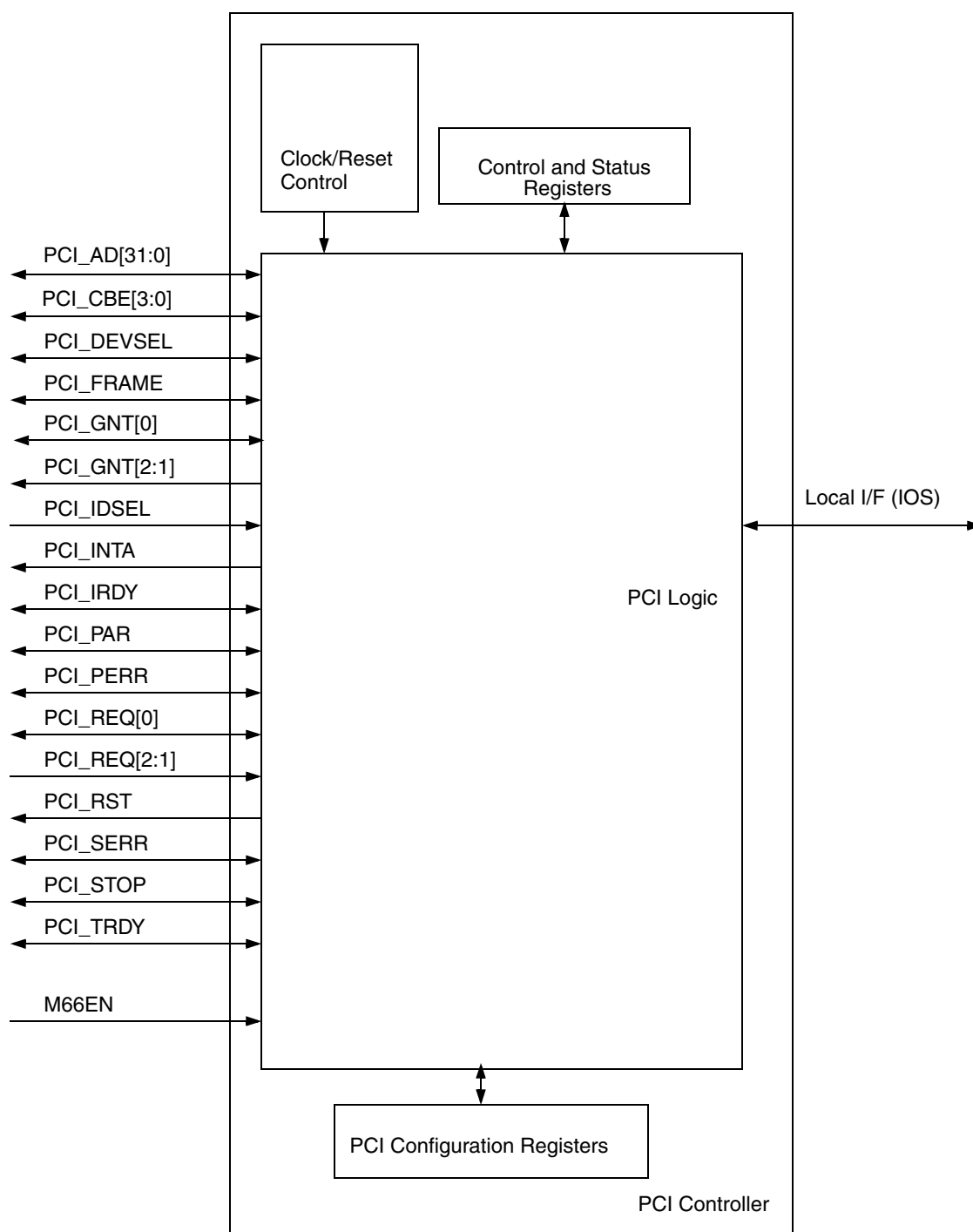


Figure 28-1. PCI Block Diagram

28.2 External Signal Description

Table 28-1 shows the properties of the external PCI signals.

Table 28-1. External Signal Properties

Name	Function	I/O	Reset State
PCI_AD[31:0]	PCI Address/Data	I/O	Not driven
PCI_CBE[3:0]	PCI Bus Command/Byte Enable	I/O	Not driven
$\overline{\text{PCI_DEVSEL}}$	PCI Device Select	I/O	Not driven
$\overline{\text{PCI_FRAME}}$	PCI Cycle Frame	I/O	Not driven
$\overline{\text{PCI_GNT}}[2:0]$	PCI Arbiter Grants	Configuration dependent	Configuration dependent
PCI_IDSEL	PCI Initialization Device Select	I	—
$\overline{\text{PCI_INTA}}$	PCI Interrupt A	O	Not driven
$\overline{\text{PCI_IRDY}}$	PCI Initiator Ready	I/O	Not driven
PCI_PAR	PCI Parity	I/O	Not driven
$\overline{\text{PCI_PERR}}$	PCI Parity Error	I/O	Not driven
$\overline{\text{PCI_REQ}}[2:0]$	PCI Arbiter Requests	Configuration dependent	Configuration dependent
$\overline{\text{PCI_RST}}$	PCI Reset	O	0
$\overline{\text{PCI_SERR}}$	PCI System Error	I/O	Not driven
$\overline{\text{PCI_STOP}}$	PCI Stop	I/O	Not driven
$\overline{\text{PCI_TRDY}}$	PCI Target Ready	I/O	Not driven

28.2.1 Detailed Signal Descriptions

Table 28-2 contains detailed descriptions of the external PCI interface signals.

Table 28-2. PCI Interface Signals—Detailed Signal Descriptions (Sheet 1 of 5)

Signal	I/O	Description
PCI_AD[31:0]	I/O	PCI Address/Data Bus. During an address phase, these signals contain a physical address. During a data phase, these signals contain the data bytes.
	O	As outputs for the bi-directional PCI address/data bus, these signals operate as described below.
		State Meaning Asserted/Negated—Represents the physical address during the address phase of a PCI transaction. During the data phase(s) of a PCI transaction, the PCI address/data bus contain the data being written. The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] define the MSB.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As inputs for the bi-directional PCI address/data bus, these signals operate as described below.
		State Meaning Asserted/Negated—Represents the address to be decoded as a check for device select during the address phase of a PCI transaction or the data being received during the data phase(s) of a PCI transaction. The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] define the MSB.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
PCI_CBE[3:0]	I/O	PCI Command/Byte Enable.
	O	As outputs for the bi-directional command/byte enable, these signals operate as described below.
		State Meaning Asserted/Negated—During the address phase, PCI_CBE[3:0] define the bus command. Byte enables determine which byte lanes carry meaningful data for PCI bus data phases. The PCI_CBE[0] signal applies to the LSB.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As inputs for the bi-directional command/byte enable, these signals operate as described below.
		State Meaning Asserted/Negated—During the address phase, PCI_CBE[3:0] indicate the command that another master is sending. During the PCI bus data phase, PCI_CBE[3:0] indicate which byte lanes are valid.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
PCI_DEVSEL	I/O	PCI Device Select.
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that this PCI controller has decoded the address and is the target of the current access. Negated—Indicates that this PCI controller has decoded the address and is not the target of the current access.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.
		State Meaning Asserted—Indicates that some PCI agent (other than this PCI controller) has decoded its address as the target of the current access. Negated—Indicates that no PCI agent has been selected.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3

Table 28-2. PCI Interface Signals—Detailed Signal Descriptions (Sheet 2 of 5)

Signal	I/O	Description	
$\overline{\text{PCI_FRAME}}$	I/O	PCI Cycle Frame. This signal is used by the current PCI master to indicate the beginning and duration of an access.	
	O	As an output, this signal operates as described below.	
		State Meaning	Asserted—Indicates that this PCI controller, acting as a PCI master, is initiating a bus transaction. While $\overline{\text{PCI_FRAME}}$ is asserted, data transfers may continue. Negated—If $\overline{\text{PCI_IRDY}}$ is asserted, indicates that the PCI transaction is in the final data phase; if $\overline{\text{PCI_IRDY}}$ is negated, indicates that the PCI bus is idle.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.	
		State Meaning	Asserted—Indicates that another PCI master is initiating a bus transaction. Negated—Indicates that the transaction is in the final data phase or that the bus is idle.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_GNT}}[0]$	I/O	PCI Arbiter Grants. Output signal on this PCI controller when the arbiter is enabled. Input signal when the arbiter is disabled. $\overline{\text{PCI_GNT}}[0]$ is a point-to-point signal. Every master has its own bus grant signal.	
	O	As an output, this signal operates as described below.	
		State Meaning	Asserted—Indicates that this PCI controller granted control of the PCI bus to agent 0. Negated—Indicates that this PCI controller did not grant control of the PCI bus to agent 0.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.	
		State Meaning	Asserted—Indicates that this PCI controller has been granted control of the PCI bus by an external arbiter. Negated—Indicates that this PCI controller has not been granted control of the PCI bus by an external arbiter.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_GNT}}[2:1]$	O	PCI Arbiter Grants. Output signals on this PCI controller when the arbiter is enabled. $\overline{\text{PCI_GNT}}[n]$ is a point-to-point signal. Every master has its own bus grant signal.	
		State Meaning	Asserted—Indicates that this PCI controller granted control of the PCI bus to agent <i>n</i> . Negated—Indicates that this PCI controller did not grant control of the PCI bus to agent <i>n</i> .
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
PCI_IDSEL	I	PCI Initialization Device Select. Used as a chip select during a PCI configuration cycle in agent mode. This signal should be tied low in host mode.	
		State Meaning	Asserted—Indicates this PCI controller is being selected as a target of a configuration read or write transactions. Negated—Indicates this PCI controller is not being selected as a target of configuration read or write transactions.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3

Table 28-2. PCI Interface Signals—Detailed Signal Descriptions (Sheet 3 of 5)

Signal	I/O	Description	
$\overline{\text{PCI_INTA}}$	O	PCI Interrupt A. This signal operates as described below.	
		State Meaning	Asserted—Indicates that the PCI controller signals an interrupt to the PCI host. Negated—Indicates that the PCI controller is not currently signalling an interrupt.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_IRDY}}$	I/O	PCI Initiator Ready. This signal is driven by the PCI when it is the initiator of a PCI transfer.	
		O As an output, this signal operates as described below.	
		State Meaning	Asserted—Indicates that this PCI controller, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, this PCI controller asserts $\overline{\text{PCI_IRDY}}$ to indicate that valid data is present on $\text{PCI_AD}[31:0]$. During a read, this PCI controller asserts $\overline{\text{PCI_IRDY}}$ to indicate it is prepared to accept data. Negated—Indicates that the PCI target needs to wait before this PCI controller, acting as a PCI master, can complete the current data phase. During a write, this PCI controller negates $\overline{\text{PCI_IRDY}}$ to insert a wait cycle when it cannot provide valid data to the target. During a read, this PCI controller negates $\overline{\text{PCI_IRDY}}$ to insert a wait cycle when it cannot accept data from the target.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.	
		State Meaning	Asserted—Indicates another PCI master can complete the current data phase of a transaction. Negated—If $\overline{\text{PCI_FRAME}}$ is asserted, indicates a wait cycle from another master. If $\overline{\text{PCI_FRAME}}$ is negated, indicates the PCI bus is idle.
PCI_PAR	I/O	PCI Parity.	
		O As output, this signal operates as described below.	
		State Meaning	Asserted—Indicates odd parity across $\text{PCI_AD}[31:0]$ and $\text{PCI_CBE}[3:0]$ during address and data phases. Negated—Indicates even parity across $\text{PCI_AD}[31:0]$ and $\text{PCI_AD}[31:0]$ during address and data phases.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As inputs for the bi-directional PCI parity, these signals operate as described below.	
		State Meaning	Asserted—Indicates odd parity driven by another PCI master or the PCI target during address and data phases. Negated—Indicates even parity driven by another PCI master or the PCI target during address and data phases.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3

Table 28-2. PCI Interface Signals—Detailed Signal Descriptions (Sheet 4 of 5)

Signal	I/O	Description
$\overline{\text{PCI_PERR}}$	I/O	PCI Parity Error
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that this PCI controller, acting as a PCI agent, detected a data parity error. (driven by the PCI initiator on reads; driven by the PCI target on writes.) Negated—Indicates no error.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.
		State Meaning Asserted—Indicates that another PCI agent detected a data parity error while this PCI controller was sourcing data (this PCI controller was acting as the PCI initiator during a write, or was acting as the PCI target during a read). Negated—Indicates no error.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_REQ}}[0]$	I/O	PCI Bus Request. Input signal on this PCI controller when the arbiter is enabled. Output signal when the arbiter is disabled. $\overline{\text{PCI_REQ}}[0]$ is a point-to-point signal. Every master has its own bus request signal.
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that the PCI is requesting control of the PCI bus to perform a transaction. Negated—Indicates that the PCI does not require use of the PCI bus.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.
		State Meaning Asserted—Indicates that agent 0 is requesting control of the PCI bus to perform a transaction. Negated—Indicates that agent 0 does not require use of the PCI bus.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_REQ}}[2:1]$	I	PCI Bus Request. Input signals on this PCI controller when the arbiter is enabled. $\overline{\text{PCI_REQ}}[n]$ is a point-to-point signal. Every master has its own bus request signal. Following is the state meaning for the $\overline{\text{PCI_REQ}}[n]$ input.
		State Meaning Asserted—Indicates that agent n is requesting control of the PCI bus to perform a transaction. Negated—Indicates that agent n does not require use of the PCI bus.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_RST}}$	O	PCI Reset. This signal is used only in Host Mode. It should be left unconnected in Agent Mode.
		State Meaning Asserted—Devices on the PCI bus are in reset. Negated—Devices on the PCI bus operate normally.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3

Table 28-2. PCI Interface Signals—Detailed Signal Descriptions (Sheet 5 of 5)

Signal	I/O	Description
$\overline{\text{PCI_SERR}}$	I/O	PCI System Error
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that an address parity error, a target-abort (when this PCI controller is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected. Negated—Indicates no error.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As inputs for the bi-directional PCI system error, these signals operate as described below.
		State Meaning Asserted—Indicates that a device (other than this PCI controller) has detected a catastrophic error. Negated—Indicates no error.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_STOP}}$	I/O	PCI Stop.
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that this PCI controller, acting as a PCI target, is requesting that the initiator stop the current transaction. Negated—Indicates that the current transaction can continue.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As inputs for the bi-directional stop, these signals operate as described below.
		State Meaning Asserted—Indicates that a target is requesting that this PCI, as the initiator, stop the current transaction. Negated—Indicates that the current transaction can continue.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
$\overline{\text{PCI_TRDY}}$	I/O	PCI Target Ready.
	O	As an output, this signal operates as described below.
		State Meaning Asserted—Indicates that this PCI controller, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, this PCI controller asserts $\overline{\text{PCI_TRDY}}$ to indicate that valid data is present on $\text{PCI_AD}[31:0]$. During a write, this PCI controller asserts $\overline{\text{PCI_TRDY}}$ to indicate that it is prepared to accept data. Negated—Indicates that the PCI initiator needs to wait before this PCI controller, acting as a PCI target, can complete the current data phase. During a read, this PCI controller negates $\overline{\text{PCI_TRDY}}$ to insert a wait cycle when it cannot provide valid data to the initiator. During a write, this PCI controller negates $\overline{\text{PCI_TRDY}}$ to insert a wait cycle when it cannot accept data from the initiator.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3
	I	As an input, this signal operates as described below.
		State Meaning Asserted—Another PCI target is able to complete the current data phase of a transaction. Negated—Indicates a wait cycle from another target.
		Timing Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.3

28.3 Memory Map and Register Definition

The PCI has three types of registers: control and status registers, PCI configuration registers, and software configuration registers. [Table 28-3](#) contains control and status registers memory-mapped on the local bus. They can be accessed by PCI masters via the PCI to the local bus via the PCI inbound memory mapped registers (PIMMR) inbound window. [Table 28-4](#) contains the PCI configuration registers mapped in the PCI configuration space. These registers are accessed by PCI masters using configuration accesses. Some fields are common to registers in both spaces to ensure consistency. These fields are discussed in the register definitions.

The third type of register is used for generating PCI configuration accesses from the local bus. These registers, listed in [Table 28-5](#), are memory-mapped on the local bus and accessed via the PIMMR window.

Table 28-3. PCI CSR Memory Map

Offset	Register	Access	Section/Page
General Registers			
0x00	PCI Error Status Register (PCI_ESR)	R/W1 to clear	28.3.1.1.1/28-11
0x04	PCI Error Capture Disable Register (PCI_ECDR)	R/W	28.3.1.1.2/28-12
0x08	PCI Error Enable Register (PCI_EER)	R/W	28.3.1.1.3/28-13
0x0C	PCI Error Attributes Capture Register (PCI_EATCR)	R/W	28.3.1.1.4/28-14
0x10	PCI Error Address Capture Register (PCI_EACR)	R/W	28.3.1.1.5/28-16
0x14	PCI Error Extended Address Capture Register (PCI_EEACR)	R/W	28.3.1.1.6/28-16
0x18	PCI Error Data Capture Register (PCI_EDCR)	R/W	28.3.1.1.7/28-17
0x20	PCI General Control Register (PCI_GCR)	R/W	28.3.1.1.8/28-18
0x24	PCI Error Control Register (PCI_ECR)	R/W	28.3.1.1.9/28-19
0x38	PCI Inbound Translation Address Register 2 (PITAR2)	R/W	28.3.1.1.11/28-21
0x40	PCI Inbound Base Address Register 2 (PIBAR2)	R/W	28.3.1.1.12/28-22
0x44	PCI Inbound Extended Base Address Register 2 (PIEBAR2)	R/W	28.3.1.1.12/28-22
0x48	PCI Inbound Window Attributes Register 2 (PIWAR2)	R/W	28.3.1.1.14/28-24
0x50	PCI Inbound Translation Address Register 1 (PITAR1)	R/W	28.3.1.1.11/28-21
0x58	PCI Inbound Base Address Register 1 (PIBAR1)	R/W	28.3.1.1.12/28-22
0x5C	PCI Inbound Extended Base Address Register 1 (PIEBAR1)	R/W	28.3.1.1.12/28-22
0x60	PCI Inbound Window Attributes Register 1 (PIWAR1)	R/W	28.3.1.1.14/28-24
0x68	PCI Inbound Translation Address Register 0 (PITAR0)	R/W	28.3.1.1.11/28-21
0x70	PCI Inbound Base Address Register 0 (PIBAR0)	R/W	28.3.1.1.12/28-22
0x78	PCI Inbound Window Attributes Register 0 (PIWAR0)	R/W	28.3.1.1.14/28-24

Table 28-4. PCI Configuration Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x00	Vendor ID	R	28.3.1.2.1/28-26
0x02	Device ID	R	28.3.1.2.2/28-27
0x04	PCI Command	R/W	28.3.1.2.3/28-28
0x06	PCI Status	Read/bit-reset	28.3.1.2.4/28-29
0x08	Revision ID	R	28.3.1.2.5/28-30
0x09	Standard Programming Interface	R	28.3.1.2.6/28-30
0x0A	Subclass Code	R	28.3.1.2.7/28-31
0x0B	Base Class Code	R	28.3.1.2.8/28-31
0x0C	Cache Line Size	R/W	28.3.1.2.9/28-32
0x0D	Latency Timer	R/W	28.3.1.2.10/28-32
0x0E	Header Type	R	28.3.1.2.11/28-33
0x0F	BIST Control	R	28.3.1.2.12/28-33
0x10	PIMMR Base Address Register	R/W	28.3.1.2.13/28-34
0x14	GPL Base Address Register 0	R/W	28.3.1.2.14/28-35
0x18	GPL Base Address Register 1	R/W	28.3.1.2.15/28-36
0x1C	GPL Extended Base Address Register 1	R/W	28.3.1.2.15/28-36
0x20	GPL Base Address Register 2	R/W	28.3.1.2.15/28-36
0x24	GPL Extended Base Address Register 2	R/W	28.3.1.2.15/28-36
0x2C	Sub System Vendor ID	R/W	28.3.1.2.16/28-37
0x2E	Sub System Device ID	R/W	28.3.1.2.17/28-37
0x34	Capabilities Pointer	R	28.3.1.2.18/28-38
0x3C	Interrupt Line	R/W	28.3.1.2.19/28-38
0x3D	Interrupt Pin	R	28.3.1.2.20/28-38
0x3E	MIN GNT	R	28.3.1.2.21/28-39
0x3F	MAX LAT	R	28.3.1.2.22/28-39
0x44	PCI Function	R/W	28.3.1.2.23/28-40
0x46	PCI Arbiter Control Register	R/W	28.3.1.2.24/28-41

Table 28-5. PCI Software Configuration Registers Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x0	CONFIG_ADDRESS	W	28.3.1.3.1/28-42
0x4	CONFIG_DATA	R/W	28.3.1.3.1/28-42
0x8	PCI Interrupt Acknowledge Register (PCI_INT_ACK)	R	28.3.1.3.2/28-44

28.3.1 Register Descriptions

28.3.1.1 Control and Status Registers

This section describes the control and status registers.

28.3.1.1.1 PCI Error Status Register (PCI_ESR)

The PCI error status register (PCI_ESR) contains status bits for various types of error conditions captured by the PCI. Each status bit is set when the corresponding error condition is captured. PCI_ESR is a write-1-to-clear type register. A bit is cleared when the register is written, and the data in the corresponding bit location is a 1. [Figure 28-2](#) shows the PCI_ESR fields.

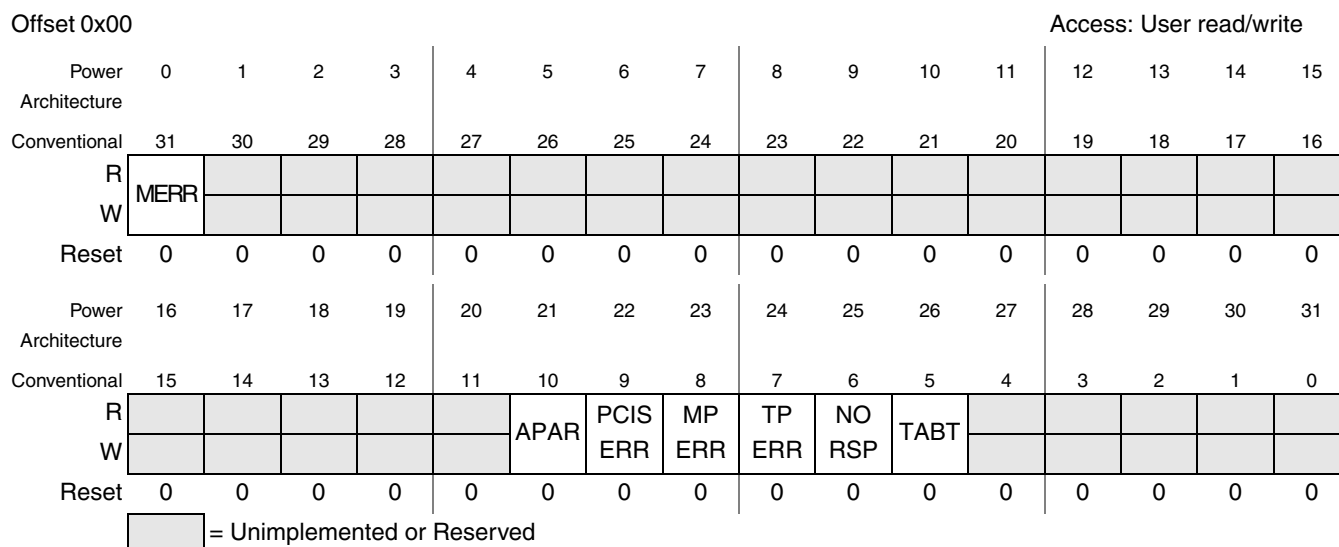


Figure 28-2. PCI Error Status Register (PCI_ESR)

Table 28-6. PCI_ESR Field Descriptions

Field	Description
MERR	Multiple Errors. This bit is set if any other bit of this register is 1 and the same error type occurs again.
APAR	Address Parity Error. This bit is set when there is an address parity error on a PCI access initiated by a device other than this PCI.
PCISERR	PCI System Error. This bit is set when the $\overline{\text{PCI_SERR}}$ input signal is asserted. See Table 28-2 for more information on $\overline{\text{PCI_SERR}}$.
MPERR	Master Parity Error. This bit is set when the $\overline{\text{PCI_PERR}}$ input signal is asserted on a write access initiated by this PCI or when a data parity error is detected by this PCI on a read access that it initiated.
TPERR	Target Parity Error. This bit is set when this PCI is the target of a transaction and the $\overline{\text{PCI_PERR}}$ input signal is asserted on a read access or a data parity error is detected by this PCI on a write access.
NORSP	No Response. This bit is set when there is no response to a transaction initiated by this PCI on the PCI bus (no PCI_DEVSEL assertion).
TABT	Target Abort. This bit is set when a PCI target abort occurs on a transaction initiated by this PCI.

28.3.1.1.2 PCI Error Capture Disable Register (PCI_ECDR)

The PCI error capture disable register (PCI_ECDR) controls the capture of the transaction that caused an error. Each bit corresponds to the error condition reported in the PCI error status register (PCI_ESR). Only the first error is captured. Disabling the capture of some error types may allow greater visibility of the significant errors.

1 = Do not capture the transaction that caused this error.

0 = Capture the transaction that caused this error.

Figure 28-3 shows the PCI_ECDR fields.

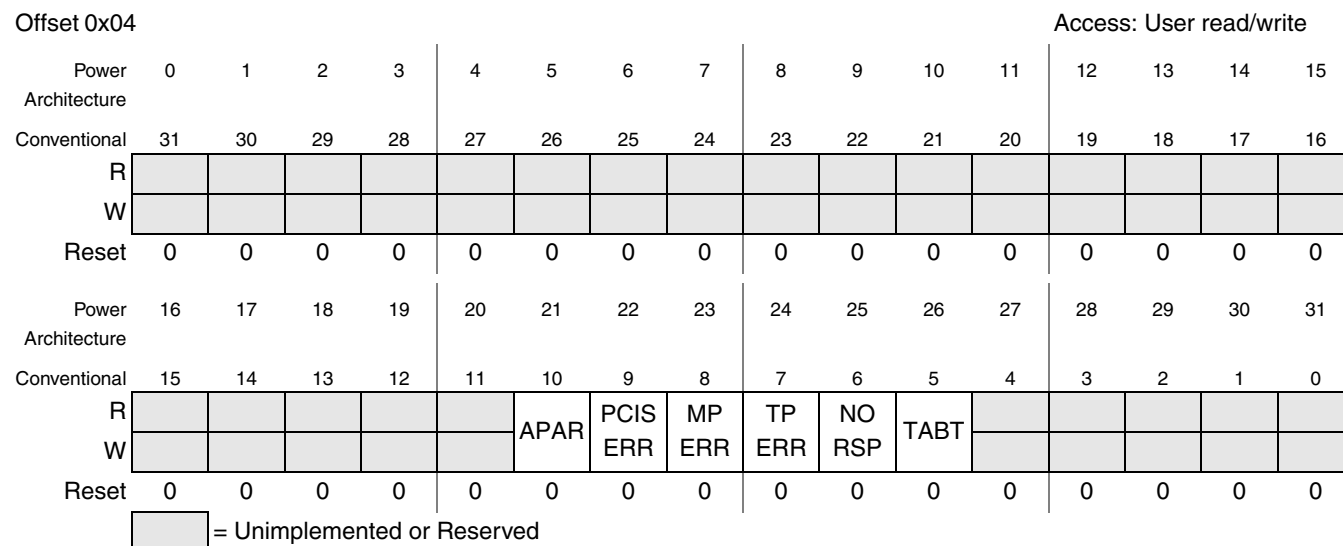


Figure 28-3. PCI Error Capture Disable Register (PCI_ECDR)

Table 28-7. PCI_ECDR Field Descriptions

Field	Description
APAR	Address Parity Error. Disable capture for address parity errors
PCISERR	PCI System Error. Disable capture for received $\overline{\text{PCI_SERR}}$ errors
MPERR	Master Parity Error. Disable capture for master $\overline{\text{PCI_PERR}}$ errors
TPERR	Target Parity Error. Disable capture for target $\overline{\text{PCI_PERR}}$ errors
NORSP	No Response. Disable capture for master-abort errors
TABT	Target Abort. Disable capture for target abort errors

28.3.1.1.3 PCI Error Enable Register (PCI_EER)

The PCI error enable register (PCI_EER) enables the assertion of an interrupt for the error conditions reported in the PCI error status register (PCI_ESR).

1 = The interrupt is enabled

0 = The interrupt is disabled

Figure 28-4 shows the PCI_EER fields.

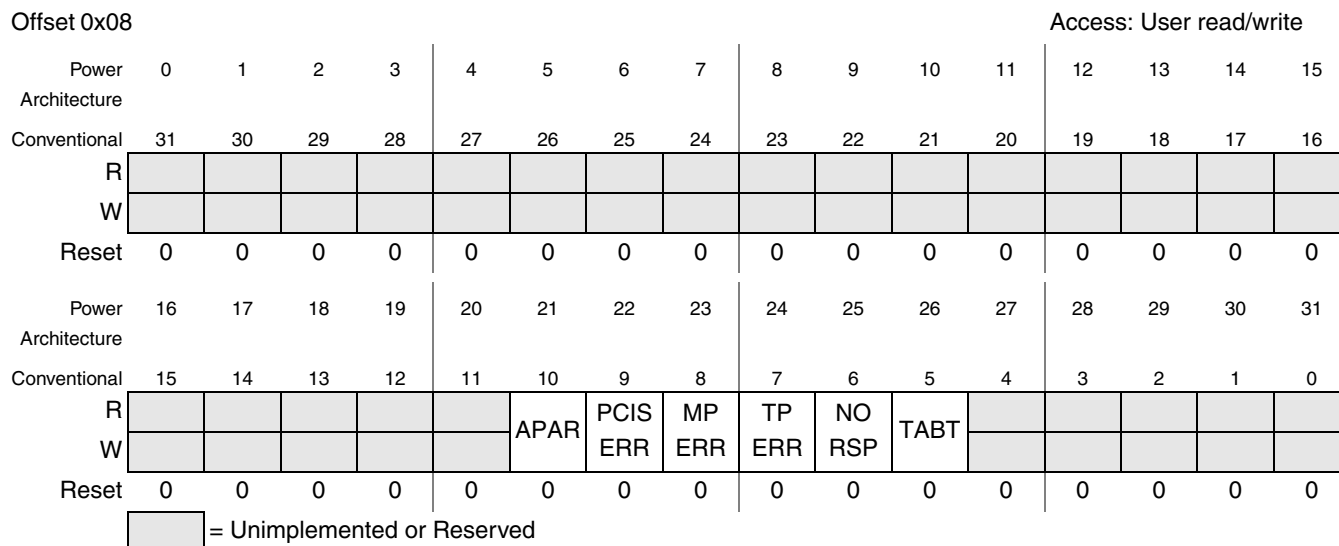


Figure 28-4. PCI Error Enable Register (PCI_ERR)

Table 28-8. PCI_ERR Field Descriptions

Field	Description
APAR	Address Parity Error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
PCISERR	PCI System Error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
MPERR	Master Parity Error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
TPERR	Target Parity Error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
NORSP	No Response. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
TABT	Target Abort. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.

28.3.1.1.4 PCI Error Attributes Capture Register (PCI_EATCR)

The PCI error attributes capture register (PCI_EATCR) stores information associated with the first PCI error captured. Figure 28-5 shows the PCI_EATCR fields.

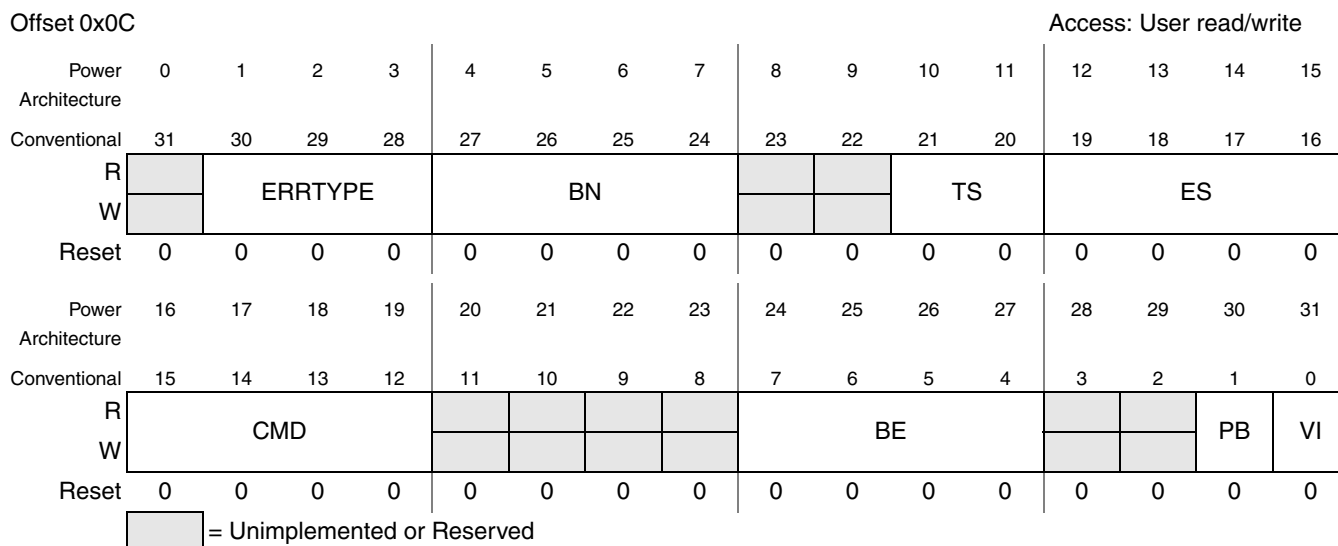


Figure 28-5. PCI Error Attributes Capture Register (PCI_EATCR)
(Register is repeated for reference.)

Table 28-9. PCI_EATCR Field Descriptions

Field	Description
ERRTYPE	First Error Type. This field is encoded to indicate the type of the first PCI error captured. 000 Address parity error 001 Write data parity error 010 Read data parity error 011 Master abort 100 Target abort 101 System error indication received 110 Parity error indication received on a read 111 Parity error indication received on a write
BN	Beat Number. This field provides the data beat number the error occurred on for data parity errors. The value of this field is undefined for other error types. The beat values are described as follows: 0000 1st beat 0001 2nd beat 0010 3rd beat 0011 4th beat 0100 5th beat 0101 6th beat 0110 7th beat 0111 8th beat 1000 9th beat or beyond (transaction larger than one cache line) Others Reserved

Offset 0x0C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		ERRTYPE			BN						TS		ES			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CMD								BE						PB	VI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 28-5. PCI Error Attributes Capture Register (PCI_EATCR)
(Register is repeated for reference.)

Table 28-9. PCI_EATCR Field Descriptions (continued)

Field	Description
TS	Transaction Size. This field contains the size of the transaction in units of doublewords (8 bytes). If the transaction crossed a cache line (32-byte) boundary, this field indicates the number of actual doublewords in the cache line the error occurred on. This field is valid only if the PCI was the master of the transaction. The size values are described as follows: 00 4 doublewords 01 1 doubleword 10 2 doublewords 11 3 doublewords
ES	Error Source. This field indicates the source of the PCI transaction. The source values are described as follows: 0000 External master 0101 DMA Others Reserved
CMD	PCI Command. This field contains the PCI command PCI_CBE[3:0] of the transaction.
BE	PCI Byte Enables. This field contains the PCI byte enables PCI_CBE[3:0] for the data word.
PB	Parity Bit. This bit contains the PCI parity bit for the captured data word.
VI	Error Information Valid. This bit indicates that the error information captured in this register, PCI_EACR, PCI_EEACR, and PCI_EDCR is valid. 0 No valid error information 1 Error information is valid

28.3.1.1.5 PCI Error Address Capture Register (PCI_EACR)

The PCI error address capture register (PCI_EACR) stores the low portion of the address associated with the first PCI error captured. [Figure 28-6](#) shows the PCI_EACR fields.

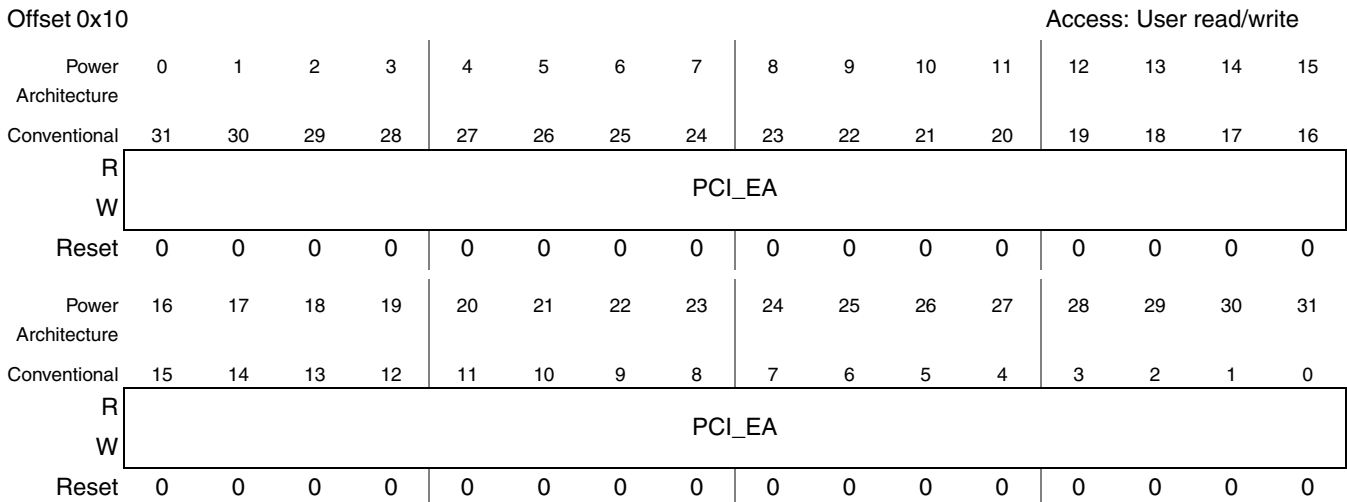


Figure 28-6. PCI Error Address Capture Register (PCI_EACR)

Table 28-10. PCI_EACR Field Descriptions

Field	Description
PCI_EA	PCI Error Address. This field contains the low portion of the address associated with the first detected error.

28.3.1.1.6 PCI Error Extended Address Capture Register (PCI_EEACR)

The PCI error extended address capture register (PCI_EEACR) stores the high portion of the address associated with the first PCI error captured. [Figure 28-7](#) shows the PCI_EEACR fields.

Offset 0x14

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PCI_EEA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PCI_EEA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. PCI Error Extended Address Capture Register (PCI_EEACR)

Table 28-11. PCI_EEACR Field Descriptions

Field	Description
PCI_EEA	PCI Error Extended Address. This field contains the high portion of the address associated with the first detected error.

28.3.1.1.7 PCI Error Data Capture Register (PCI_EDLCR)

The PCI error data capture register (PCI_EDCR) stores the data associated with the first PCI error captured. [Figure 28-8](#) shows the PCI_EDCR fields.

Offset 0x18

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PCI_EDCR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PCI_EDCR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. PCI Error Data Low Capture Register (PCI_EDCR)

Table 28-12. PCI_EDCR Field Descriptions

Field	Description
PCI_EDCR	PCI Error Data. This field contains the data associated with the first detected error.

28.3.1.1.8 PCI General Control Register (PCI_GCR)

The PCI general control register (PCI_GCR) controls the behavior of the internal arbiter, the state of the bus signals, and contains a bit for controlling the PCI reset signal when in host mode. Figure 28-9 shows the PCI_GCR fields.

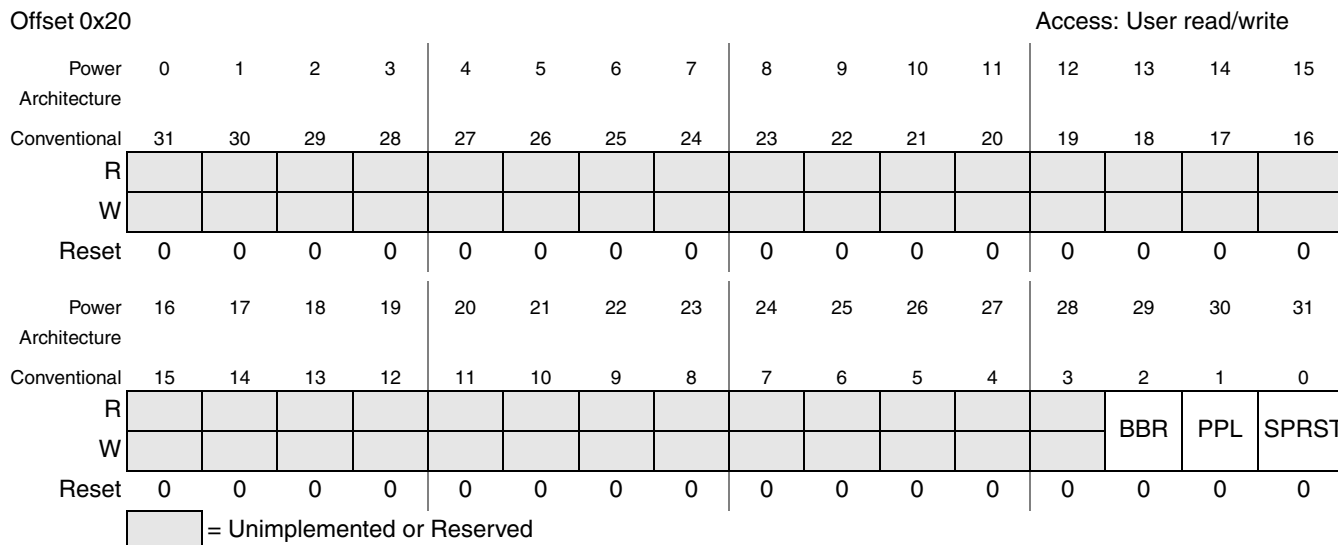


Figure 28-9. PCI General Control Register (PCI_GCR)

Table 28-13. PCI_GCR Field Descriptions

Field	Description
BBR	Block Bus Requests. When this bit is set, all bus requests from external devices to the PCI's internal arbiter are blocked and the bus is continuously granted to the PCI. This bit could be used to prepare for entering a low-power mode by preventing transactions on the PCI bus. 0 External bus requests are treated normally. 1 Block external bus requests Note: Before setting the BBR bit, parking should be changed to park to internal PCI master (set PCI_ACR[PM] = 1).
PPL	PCI Pins Low. Setting this bit forces all the output and bidirectional pins of the PCI bus to be driven low. This bit could be used to put the bus signals in a safe electrical state when the devices on the bus are powered down. This bit should never be set during normal operation of the PCI bus. 0 PCI pins function normally 1 PCI pins in the low state
SPRST	Soft PCI Reset. This bit provides software control of the $\overline{\text{PCI_RESET_OUT}}$ output signal. It is only valid in host mode. 0 $\overline{\text{PCI_RESET_OUT}}$ is driven low. 1 $\overline{\text{PCI_RESET_OUT}}$ is driven high.

28.3.1.1.9 PCI Error Control Register (PCI_ECR)

The PCI error control register (PCI_ECR) determines whether an interrupt or machine check is generated for the error conditions reported in the PCI error status register (PCI_ESR). If the corresponding bit in the PCI error enable register (PCI_EER) is clear, the bit in the PCI error control register (PCI_ECR) has no effect.

1 = A machine check is generated.

0 = An interrupt is generated.

Figure 28-10 shows the PCI_ECR fields.

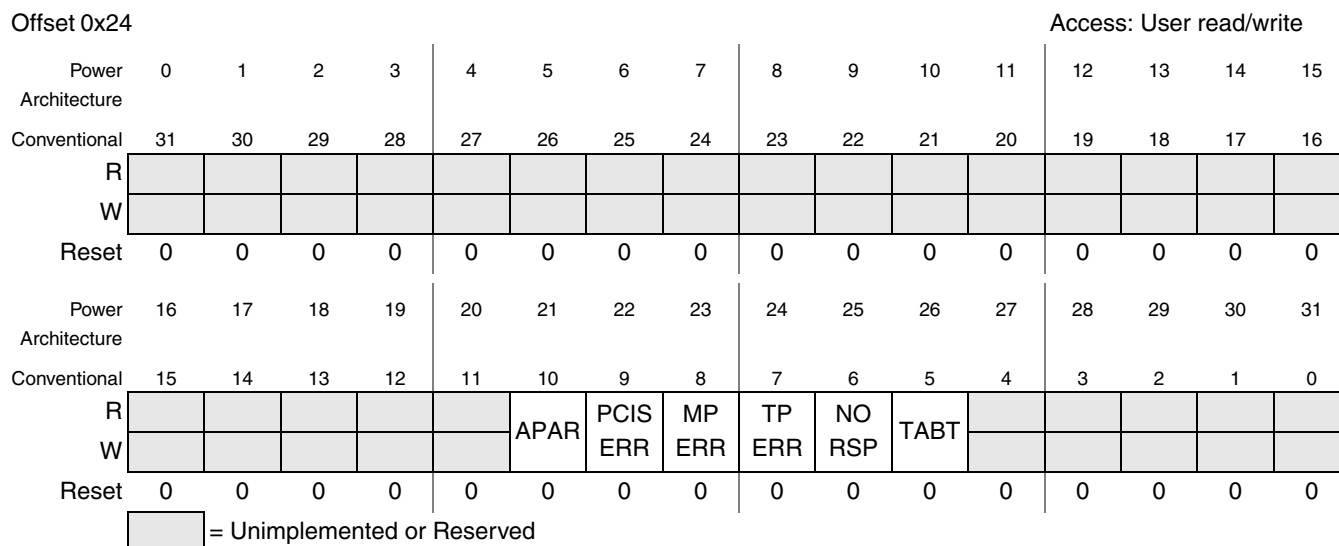


Figure 28-10. PCI Error Control Register 0 (PCI_ECR)

Table 28-14. PCI_ECR Field Descriptions

Field	Description
APAR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
PCISERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
MPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
TPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
NORSP	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
TABT	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.

28.3.1.1.10 PCI General Status Register (PCI_GSR)

The PCI general status register (PCI_GSR) provides status information.

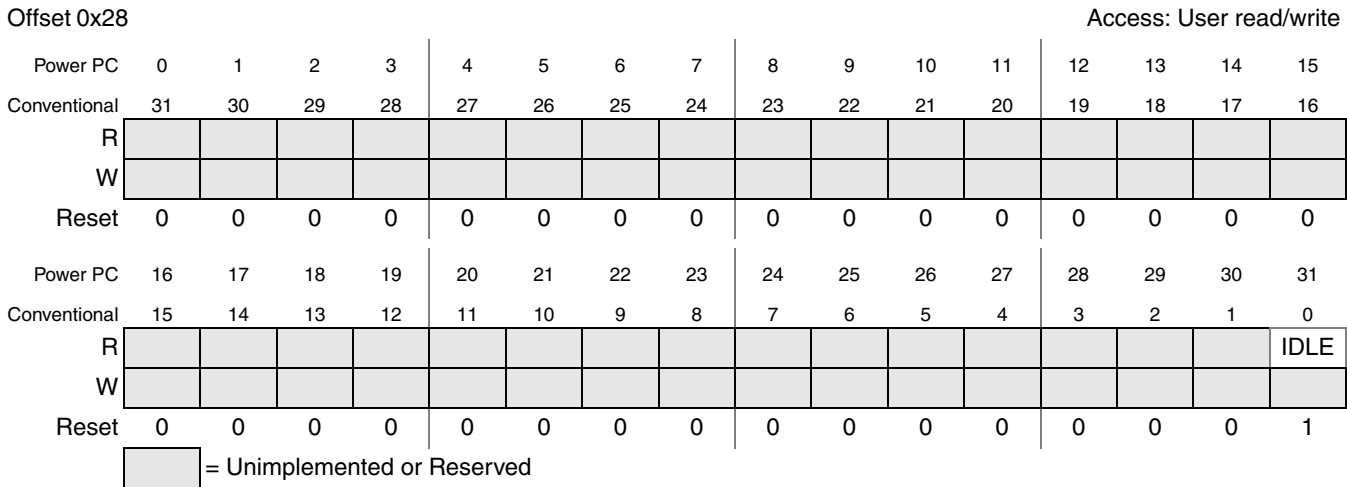


Figure 28-11. PCI General Status Register (PCI_GSR)

Table 28-15. PCI_GSR Field Descriptions

Field	Description
IDLE	PCI controller is idle. This bit could be used to determine when the PCI bus is totally idle before setting PCI_GCR[PPL]. 0 The PCI controller is active. 1 The PCI controller is idle.

28.3.1.1.11 PCI Inbound Translation Address Registers (PITAR_n)

The PCI inbound translation address registers (PITAR_n) define the starting point of the inbound translation windows in the local memory space.

Offset 0x38, 0x50, 0x68

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													TA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-12. PCI Inbound Translation Address Registers (PITAR_n)

Table 28-16. PITAR_n Field Descriptions

Field	Description
TA	Translation Address. This field contains the starting address of the inbound translated address. This 20-bit field corresponds to address bits 12-31 on the CSB bus. Window 0 does not support 64-bit address. Therefore, only the bits in the TA field define the address bit 12-31.

28.3.1.1.12 PCI Inbound Base Address Registers (PIBAR_n)

The PCI inbound base address registers (PIBAR_n) define the starting point of the inbound windows in the PCI memory space. A write to a PIBAR_n register also causes a change in the base address bits not masked by PIWAR_n in the corresponding general purpose local access (GPL) base address register in the PCI configuration space. This write operation does not change the bits masked by the IWS field. For read operation, these masked bits always return zeros. [Figure 28-13](#) shows the PIBAR_n fields.

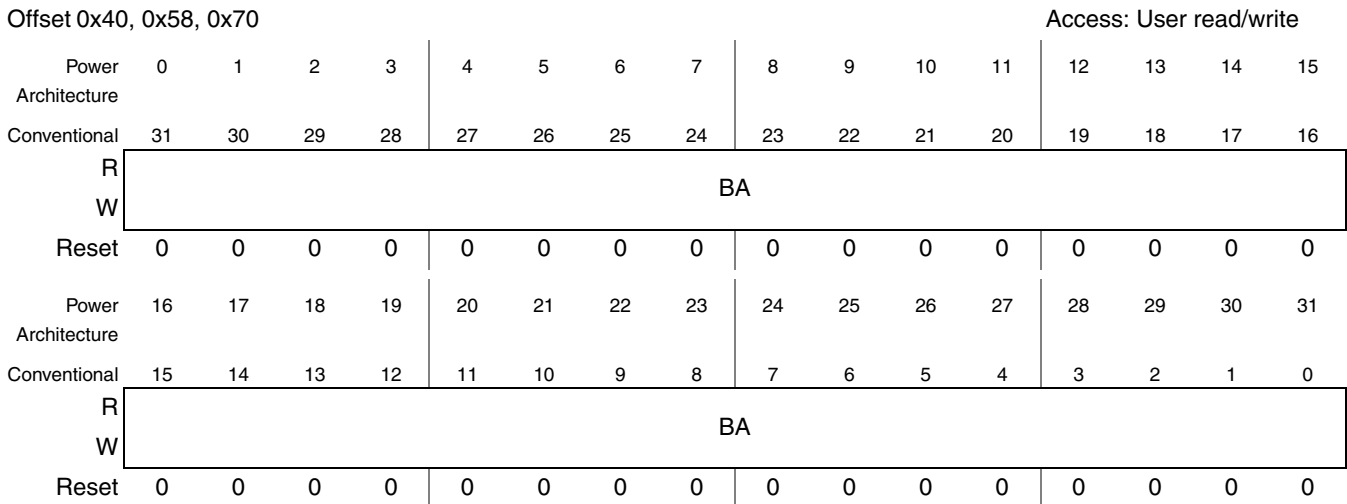


Figure 28-13. PCI Inbound Base Address Registers (PIBAR_n)

Table 28-17. PIBAR_n Field Descriptions

Field	Description
BA	Base Address. This field contains the starting address in the PCI memory space of the inbound window. This field corresponds to bits 43–12 of a 64-bit address. In PIBAR0, the upper 12 bits are reserved because only a 32-bit address is supported.

28.3.1.1.13 PCI Inbound Extended Base Address Registers (PIEBAR_n)

The PCI inbound extended base address registers (PIEBAR_n) define the high portion of the starting point of the inbound windows in the PCI memory space. Figure 28-14 shows the PIEBAR_n fields.

Offset 0x44, 0x5C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													EBA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EBA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-14. PCI Inbound Extended Base Address Registers (PIEBAR_n)

Table 28-18. PIEBAR_n Field Descriptions

Field	Description
EBA	Extended Base Address. This field contains the high portion of the starting address in the PCI memory space of the inbound base address. This 20-bit field corresponds to bits 63–44 of a 64-bit address.

28.3.1.1.14 PCI Inbound Window Attribute Registers (PIWAR_n)

The PCI inbound window attribute registers (PIWAR_n) define the size of the inbound translation window. It also defines some properties of the window. Figure 28-16 shows the PIWAR_n fields.

Offset 0x48, 0x60, 0x78

Access: User read/write

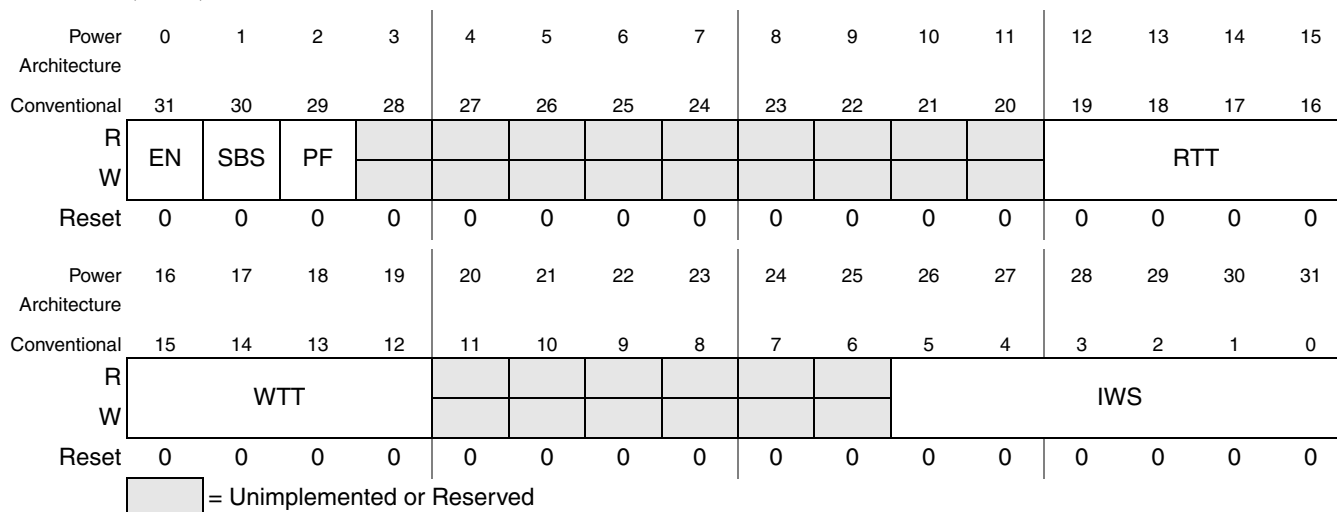


Figure 28-15. PCI Inbound Window Attribute Registers (PIWAR_n)
(Register is repeated for reference.)

Table 28-19. PIWAR_n Field Descriptions

Field	Description
EN	Enable. This bit enables the address translation window. 0 Address translation is disabled for this window. 1 Address translation is enabled for this window. PCI addresses that match the definition of the window are recognized by the PCI and translated to the local memory space.
SBS	Special Byte Swap. This bit indicates that each 32-bit word of transactions that are translated through this window should be byte-swapped. 0 No special byte swap. PCI access is little endian. 1 Special byte swap. PCI access is big endian.
PF	Prefetchable. This bit defines whether the transactions translated through this window are prefetchable on the local bus. Streaming the transactions requires the memory space to be prefetchable. 0 Not prefetchable 1 Prefetchable
RTT	Read Transaction Type. This field determines the type of transaction performed on the local bus when the PCI transaction is a read. The RTT values are described as follows: 0100 Read without snoop on system bus 0101 Read with snoop on system bus Others Reserved
WTT	Write Transaction Type. This field determines the type of transaction performed on the local bus when the PCI transaction is a write. The WTT values are described as follows: 0100 Write without snoop of local processor 0101 Write with snoop of local processor Others Reserved

Offset 0x48, 0x60, 0x78

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN	SBS	PF										RTT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WTT												IWS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 28-15. PCI Inbound Window Attribute Registers (PIWAR_n)
 (Register is repeated for reference.)

Table 28-19. PIWAR_n Field Descriptions (continued)

Field	Description
IWS	Inbound Window Size. This field determines the size of the inbound translation window. Inbound translation window size N is the encoded $2^{(N+1)}$ bytes window size. The smallest window is 4 Kbytes (N = 11). 000000–001010 Reserved 001011 4-Kbyte window size 001100 8-Kbyte window size ... 011110 2-Gbyte window size 011111–111111 Reserved

28.3.1.2 PCI Configuration Space Registers

This section describes the registers contained in the PCI configuration space. These registers are shown with descending bit numbering to correspond to the PCI standard.

28.3.1.2.1 Vendor ID Configuration Register

Figure 28-17 shows the vendor ID fields. This is a read only register.

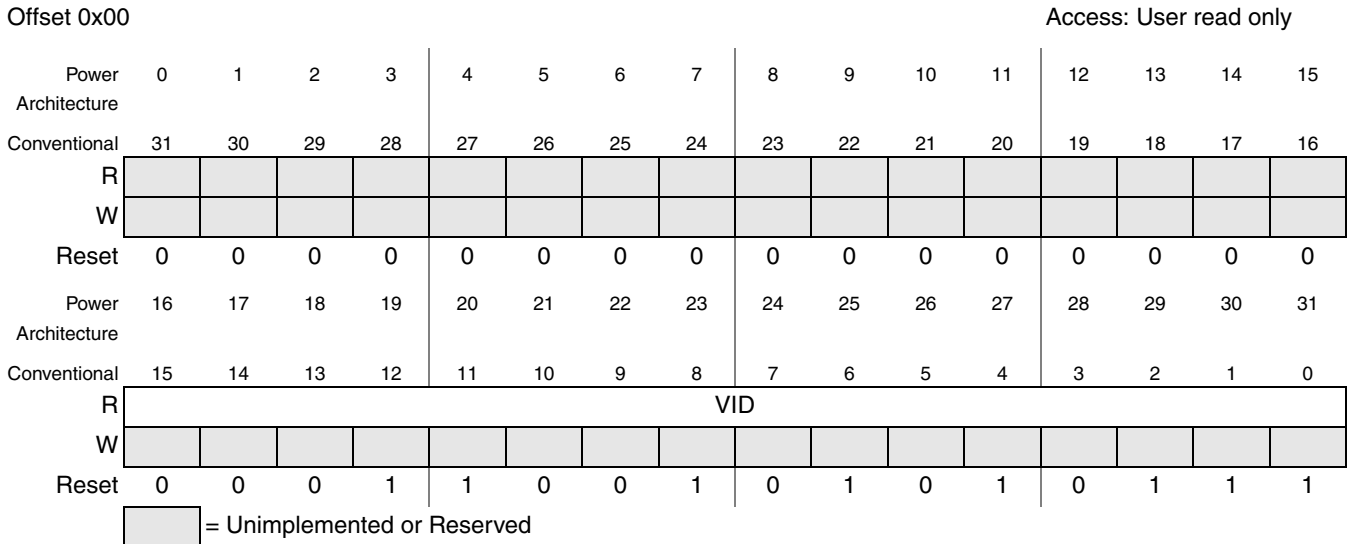


Figure 28-16. Vendor ID Configuration Register (VID)

Table 28-20. VID Field Descriptions

Field	Description
VID	Vendor ID. This field identifies the manufacturer of the device: 0x1957

28.3.1.2.2 Device ID Configuration Register

Figure 28-17 shows the device ID fields. This is a read only register.

Offset 0x02

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DID															
W																
Reset	Device Dependent															

= Unimplemented or Reserved

Figure 28-17. Device ID Configuration Register (DID)

Table 28-21. DID Field Descriptions

Field	Description
DID	Device ID. This field identifies the device. 0x580C

28.3.1.2.3 PCI Command Configuration Register

Figure 28-18 shows the PCI command fields.

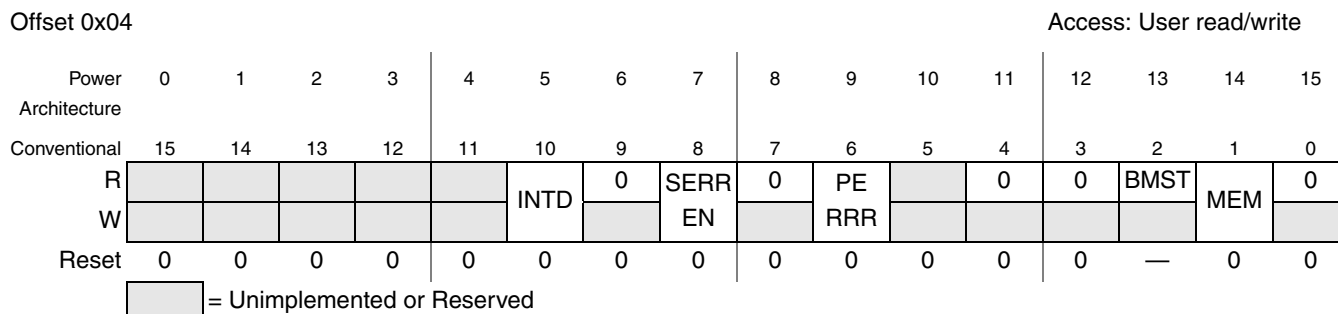


Figure 28-18. PCI Command Configuration Register

Table 28-22. PCI Command Configuration Register Field Descriptions

Field	Description
INTD	Interrupt Disable. Setting this bit masks the PCI_INTA output. 0 PCI_INTA provides the device interrupt status. 1 PCI_INTA is always negated
SERREN	SERR Enable. This bit is an enable bit for the SERR driver. Address parity errors are reported only if this bit and bit 6 are 1. 0 PCI_SERR is never asserted. 1 PCI_SERR may be asserted to indicate error conditions.
PERRR	Parity Error Response. This bit controls the PCI's response to a parity error. 0 Parity errors are ignored and normal operation continues. 1 Standard parity error treatment.
MWI (Bit 4)	Memory-Write-and-Invalidate. Hard-wired to 0.
SC (Bit 3)	Special Cycles. Hard-wired to 0.
BMST	Bus Master. This bit controls the PCI's ability to be a master on the PCI bus. At reset, this bit is set in host mode. 0 The PCI does not generate PCI accesses. 1 The PCI behaves as a bus master.
MEM	Memory Space. This bit controls the response to memory space accesses. 0 The PCI does not respond to Memory Space accesses. 1 The PCI as a target responds to Memory Space accesses.
I/O (Bit 0)	I/O space. Hard-wired to 0.

28.3.1.2.4 PCI Status Configuration Register

This register is used to record status information for PCI bus-related events. Some of the bits are hard-wired to indicate the capabilities of the PCI. Other bits can be cleared by writing 1 to the bit location. Figure 28-19 shows the PCI status fields.

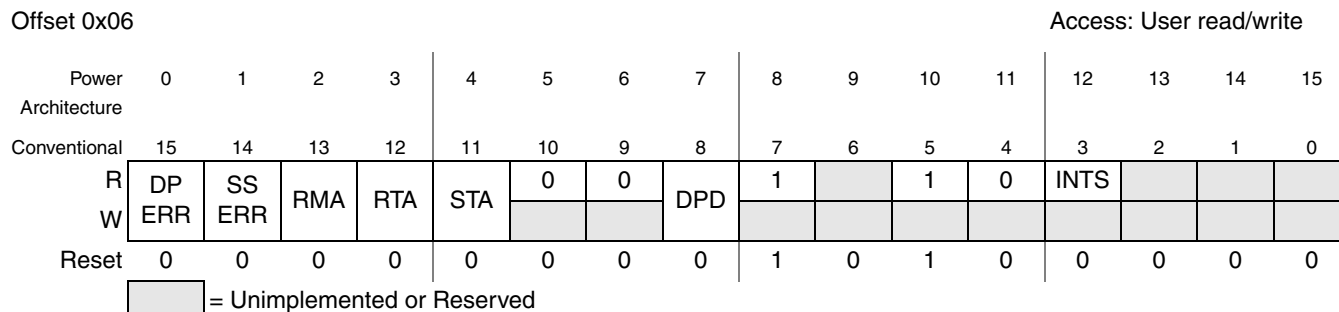


Figure 28-19. PCI Status Configuration Register

Table 28-23. PCI Status Configuration Register Field Descriptions

Field	Description
DPERR	Detected Parity Error. This bit is set when the PCI detects a parity error on the PCI bus, even if parity error handling is disabled (as controlled by bit 6 in the PCI Command register).
SSERR	Signalled System Error. This bit is set when $\overline{\text{PCI_SERR}}$ is asserted.
RMA	Received Master Abort. This bit is set when the PCI, acting as the PCI master on the PCI bus, terminates a transaction (except for a special-cycle) using master-abort.
RTA	Received Target Abort. This bit is set when a transaction initiated by this PCI on the PCI bus is terminated by a target-abort.
STA	Signalled Target Abort. This bit is set when the PCI, acting as the PCI target on the PCI bus, issues a target-abort to a PCI master.
DEVSEL_T (Bits 10–9)	DEVSEL Timing. Hard-wired to 00.
DPD	Master Data Parity Error. This bit is set when a data parity error is detected on the PCI bus, if the PCI is the master that initiated the transaction and bit 6 in the PCI command register is set.
FB–BC (Bit 7)	Fast Back-to-Back Capable. Hard-wired to 1.
66M (Bit 5)	66 MHz Capable. Hard-wired to 1.
CL (Bit 4)	Capabilities List. Hard-wired to 0.
INTS	Interrupt Status. This bit contains the status of the unmasked device interrupt. The value of this bit is not affected by the INTD bit of the PCI Command Configuration Register.

28.3.1.2.5 Revision ID Configuration Register

Figure 28-20 shows the revision ID fields.

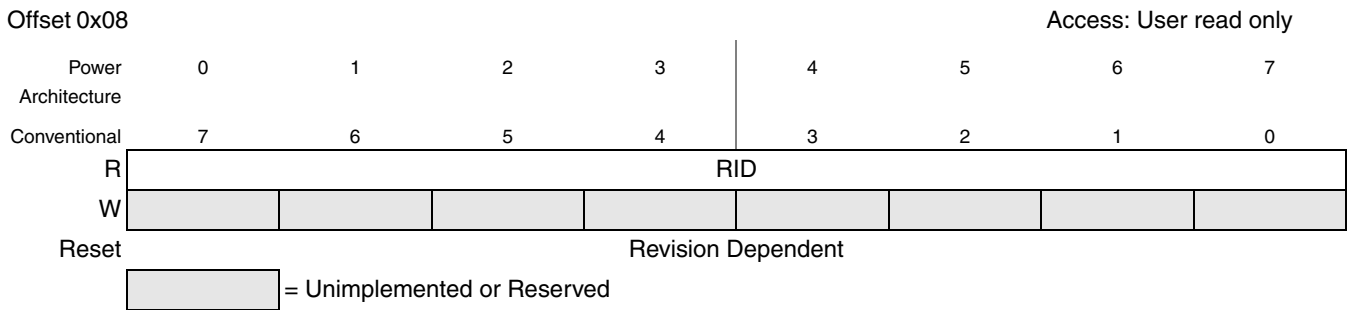


Figure 28-20. Revision ID Configuration Register

Table 28-24. Revision ID Configuration Register Field Descriptions

Field	Description
RID	Revision ID. This field specifies a revision code of the PCI. This field is hard-wired to 0x00.

28.3.1.2.6 Standard Programming Interface Configuration Register

Figure 28-21 shows the standard programming interface fields. This is the lower byte of the Class Code.

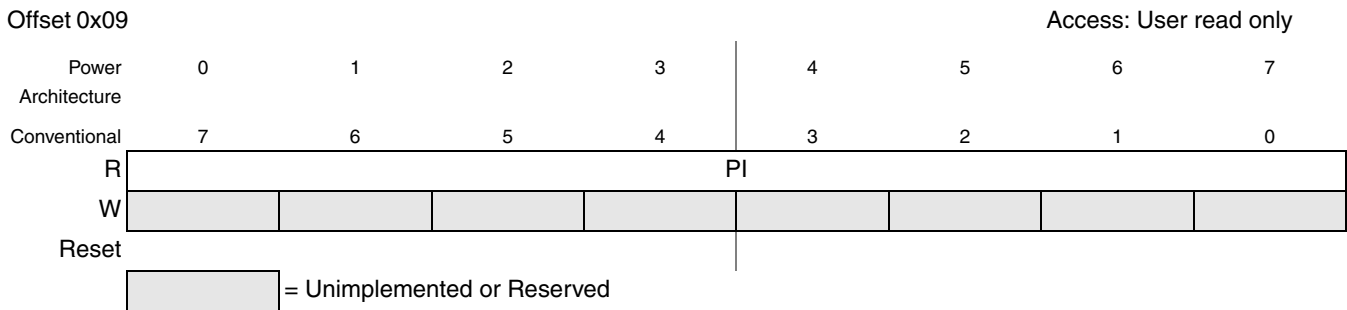


Figure 28-21. Standard Programming Interface Configuration Register

Table 28-25. Standard Programming Interface Configuration Register Field Descriptions

Field	Description
PI	Programming Interface. This field is hard-wired to 0x00.

28.3.1.2.7 Subclass Code Configuration Register

Figure 28-22 shows the subclass code fields. This is the middle byte of the class code.

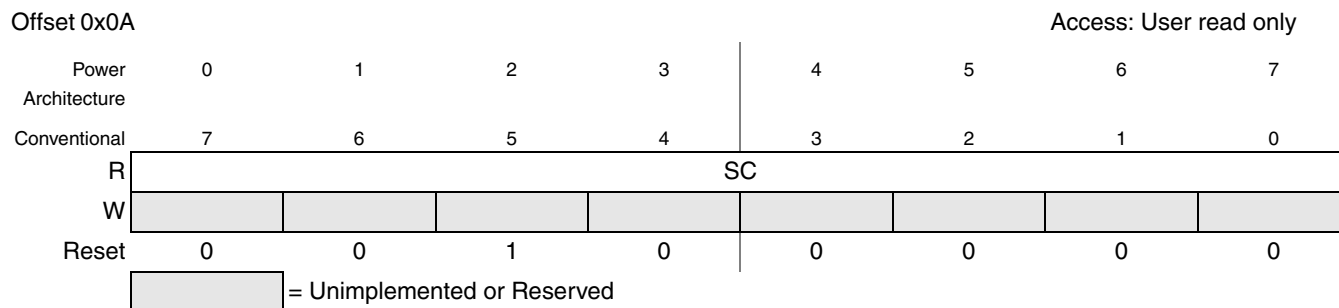


Figure 28-22. Subclass Code Configuration Register

Table 28-26. Subclass Code Configuration Register Field Descriptions

Field	Description
SC	Sub-Class Code. This field is hard-wired to 0x20, indicating a Power Architecture processor.

28.3.1.2.8 Base Class Code Configuration Register

Figure 28-23 shows the base class code fields. This is the upper byte of the Class Code.

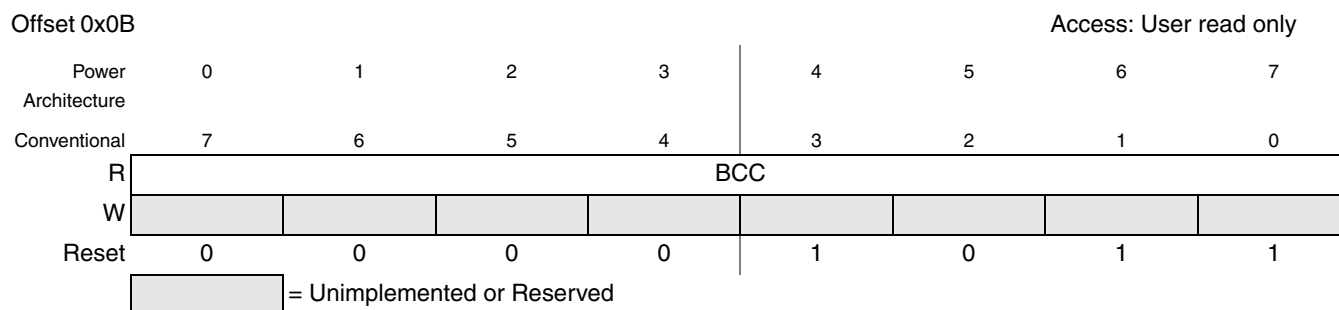


Figure 28-23. Subclass Code Configuration Register

Table 28-27. Subclass Code Configuration Register Field Descriptions

Field	Description
BCC	Base Class Code. This field is hard-wired to 0x0B, indicating a processor.

28.3.1.2.9 Cache Line Size Configuration Register

Figure 28-24 shows the cache line size fields.

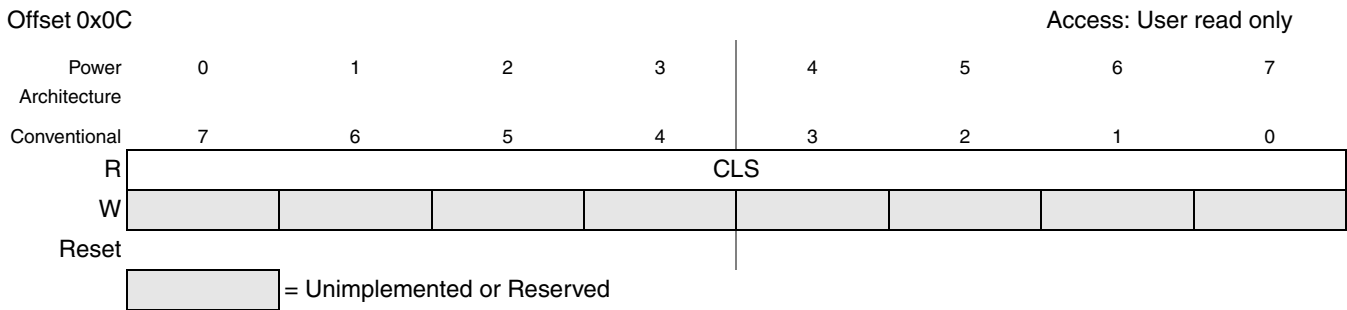


Figure 28-24. Cache Line Size Configuration Register

Table 28-28. Cache Line Size Configuration Register

Field	Description
CLS	Cache Line Size. This field represents the cache-line size of the system in terms of 32-bit words. Although the register is writable, only the value 0x08 is legal.

28.3.1.2.10 Latency Timer Configuration Register

Figure 28-25 shows the latency timer fields.

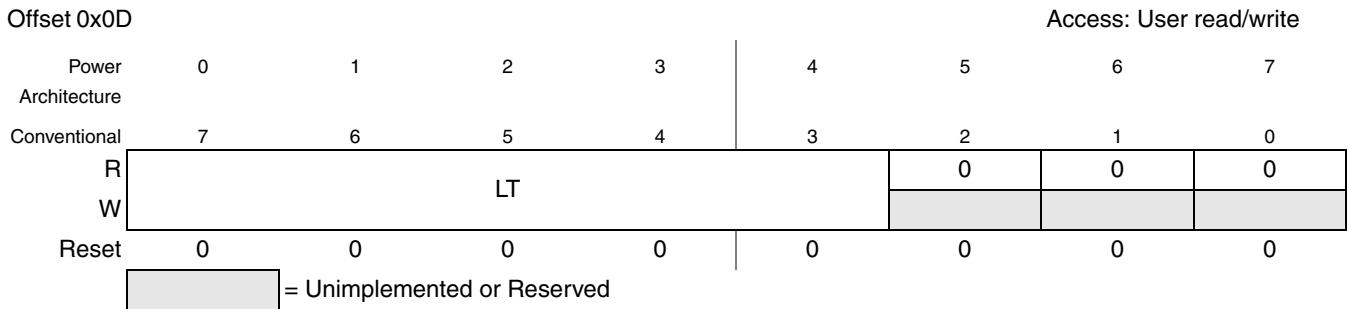


Figure 28-25. Latency Timer Configuration Register

Table 28-29. Latency Timer Configuration Register

Field	Description
LT	Latency Timer. This field specifies, with a granularity of eight PCI clocks, the length of time that the PCI, when mastering a transaction, may hold the bus as the result of a bus grant. Refer to the PCI 2.3 specification for the rules by which the PCI completes transactions when the timer has expired.

28.3.1.2.11 Header Type Configuration Register

Figure 28-26 shows the read-only header type register hard-wired to 0x00.

Offset 0x0E				Access: User read only				
Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 28-26. Header Type Configuration Register

28.3.1.2.12 BIST Control Configuration Register

Figure 28-27 shows the read-only BIST control register that is hard-wired to 0x00.

Offset 0x0F				Access: User read only				
Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 28-27. BIST Control Configuration Register

28.3.1.2.13 PIMMR Base Address Configuration Register

Figure 28-28 shows the PIMMR base address register fields.

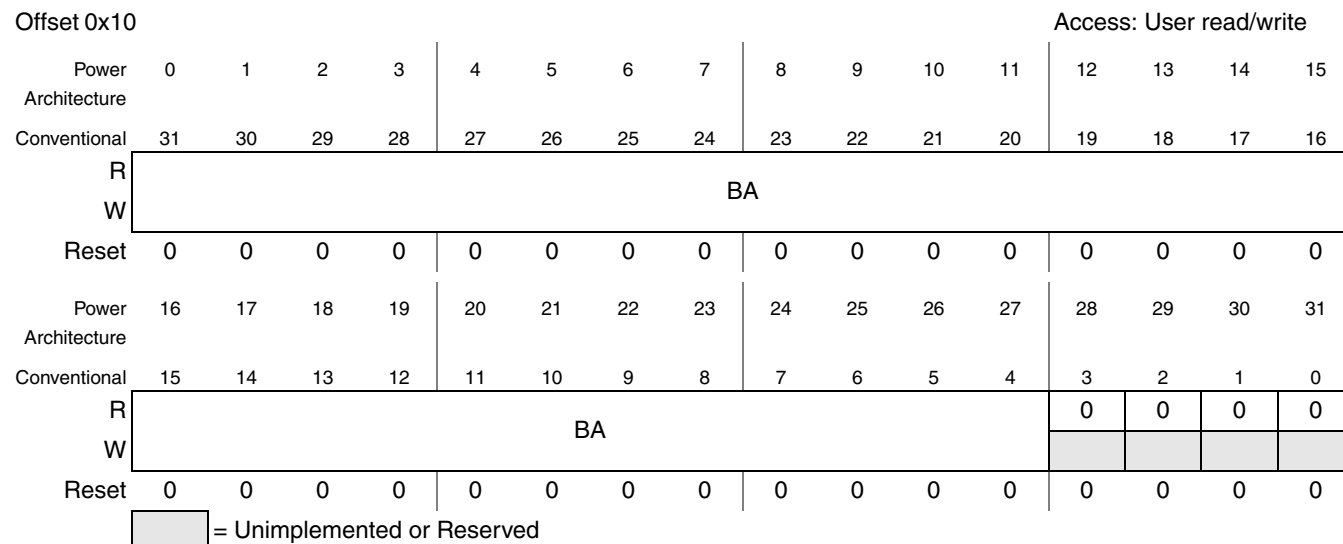


Figure 28-28. PIMMR Base Address Configuration Register

Table 28-30. PIMMR Base Address Configuration Register Field Descriptions

Field	Description
BA	Base Address. This field defines the base address for the internal (on-chip) memory-mapped register space. The size of this space is 1MB.
PRE (Bit 3)	Prefetchable. Hard-wired to 0.
T (Bits 2–1)	Type. Hard-wired to 00.
MSI (Bit 0)	Memory Space Indicator. Hard-wired to 0

28.3.1.2.14 GPL Base Address Register 0

The GPL base address register 0 is provided to allow access to local memory space. This register is closely tied to PIBAR0 and PIWAR0 in the CSR memory space. A write to GPL base address register 0 also causes a change in the base address bits not masked according to the IWS field of PIWAR0 in PIBAR0. This write operation does not change the bits masked by the IWS field. For read operation, these masked bits always return zeros. Figure 28-29 shows the GPL base address register 0 fields.

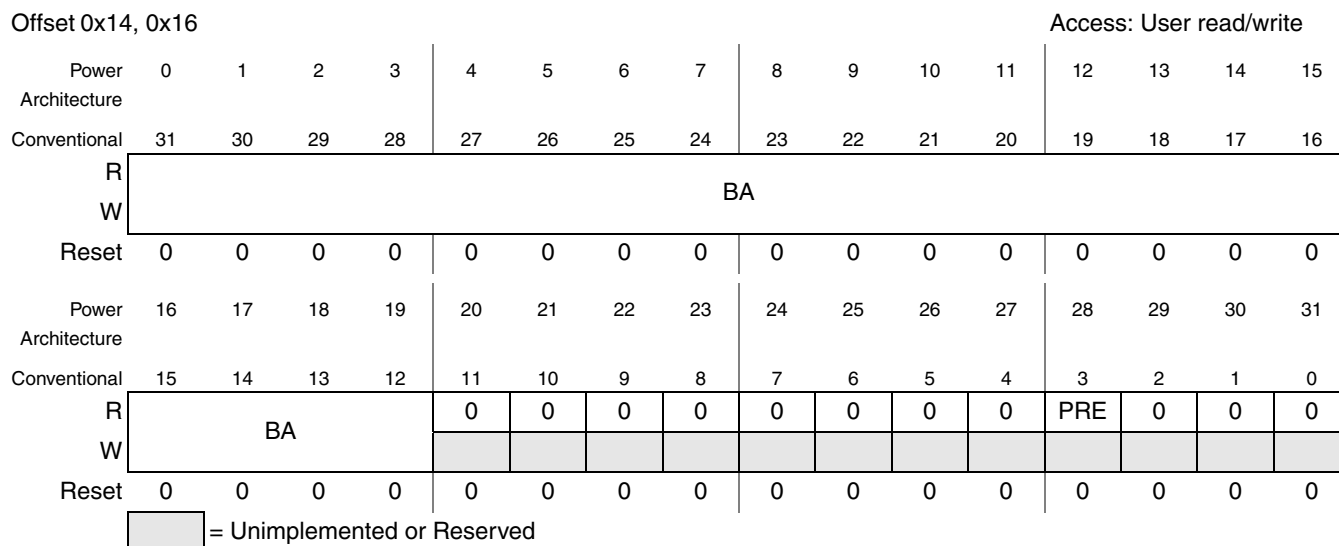


Figure 28-29. GPL Base Address Register 0

Table 28-31. GPL Base Address Register 0

Field	Description
BA	Base Address. This field defines the base address for the inbound window. Bits 11:4 are hard-wired to 0 because the minimum window size is 4KB.
PRE (Bit 3)	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR0.
T (Bits 2–1)	Type. Hard-wired to 00.
MSI (Bit 0)	Memory space indicator. Hard-wired to 0

28.3.1.2.15 GPL Base Address Registers 1,2

The general purpose local access base address registers are provided to allow access to local memory space. These registers are closely tied to PIBAR $_n$ and PIWAR $_n$ in the CSR memory space. A write to a GPL base address register also causes a change in the base address bits not masked according to the IWS field of PIWAR $_n$ in the corresponding PIBAR $_n$. This write operation does not change the bits masked by the IWS field. For read operation, these masked bits always return zeros. Figure 28-30 shows the GPL base address register 1,2 fields.

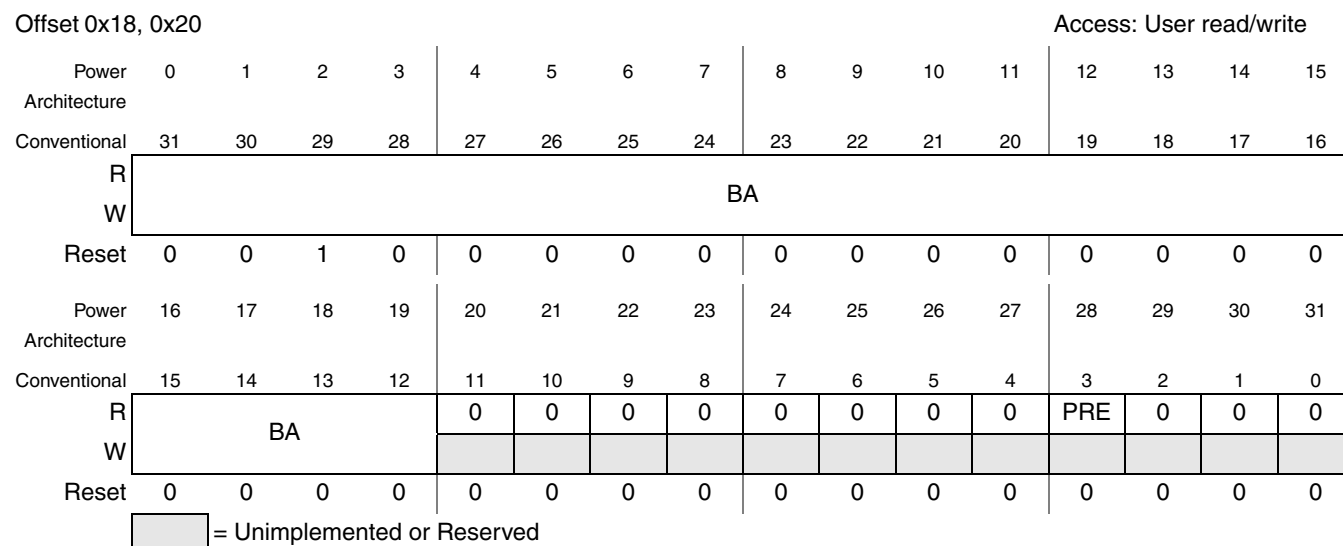


Figure 28-30. GPL Base Address Register 0

Table 28-32. GPL Base Address Register 0

Field	Description
BA	Base Address. This field defines the two portion of the base address for the inbound window. Bits 11–4 are hard-wired to 0 because the minimum window size is 4KB.
PRE (Bit 3)	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR $_n$.
T (Bits 2–1)	Type. Hard-wired to 10.
MSI (Bit 0)	Memory Space Indicator. Hard-wired to 0

28.3.1.2.16 Sub-System Vendor ID Configuration Register

Figure 28-31 shows the sub-system vendor ID fields.

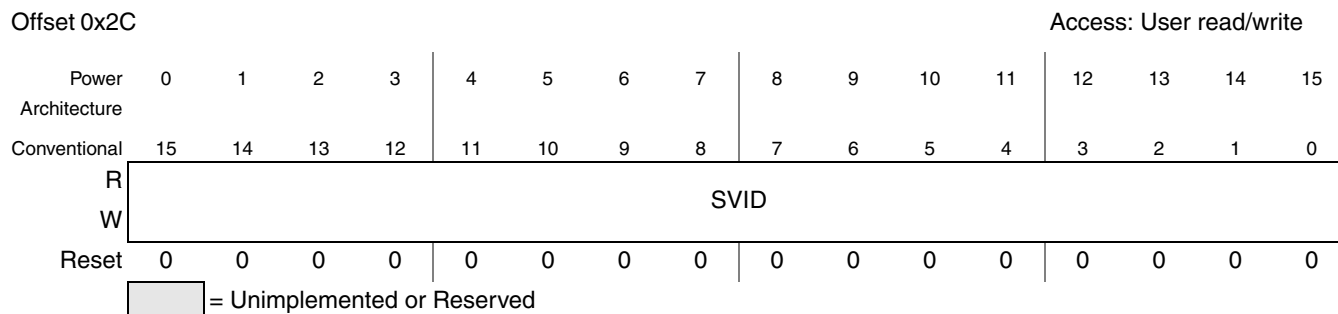


Figure 28-31. Sub-System Vendor ID Configuration Register

Table 28-33. Sub-System Vendor ID Configuration Register Field Descriptions

Field	Description
SVID	Sub-System Vendor ID. This field identifies the manufacturer of the board or sub-system that contains this device.

28.3.1.2.17 Sub-System Device ID Configuration Register

Figure 28-32 shows the sub-system device configuration register ID fields.

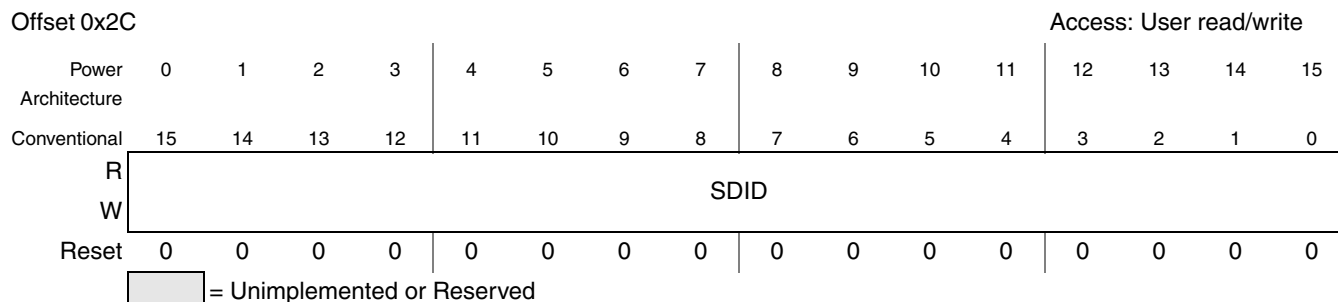


Figure 28-32. Sub-System Device ID Configuration Register

Table 28-34. Sub-System Device ID Configuration Register Field Descriptions

Field	Description
SDID	Sub-System Device ID. This field identifies the board or sub-system that contains this device.

28.3.1.2.18 Capabilities Pointer Configuration Register

The capabilities pointer register specifies the byte offset in the PCI configuration space that contains the first item in the capabilities list. [Figure 28-33](#) shows the capabilities pointer configuration register fields.

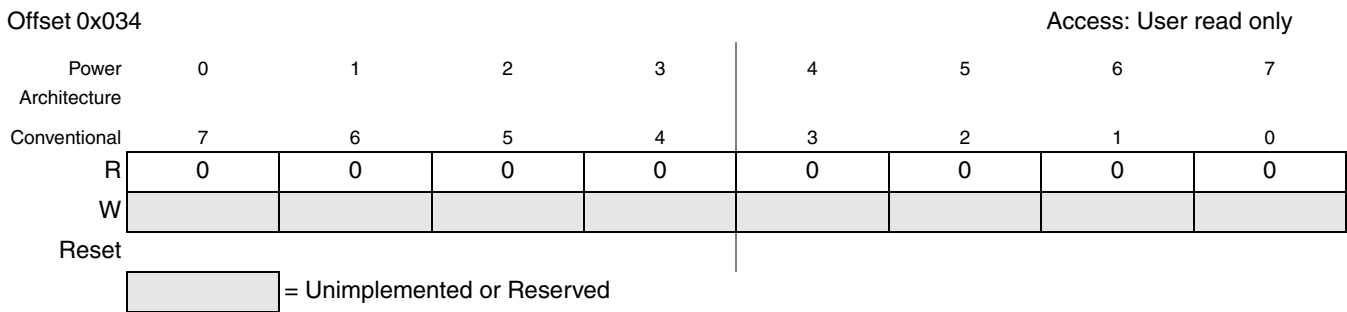


Figure 28-33. Capabilities Pointer Configuration Register

28.3.1.2.19 Interrupt Line Configuration Register

[Figure 28-34](#) shows the interrupt line configuration register fields.

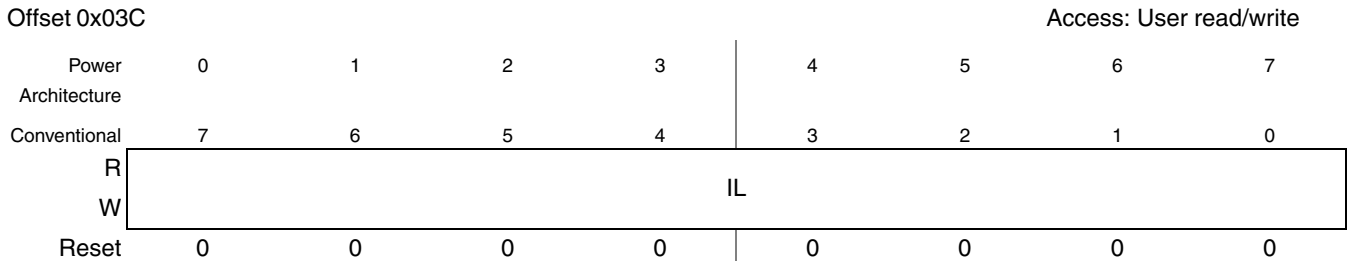


Figure 28-34. Interrupt Line Configuration Register

Table 28-35. Interrupt Line Configuration Register Field Descriptions

Field	Description
IL	Interrupt Line. This is used to communicate interrupt line routing information. The value has no effect on the operation of the PCI.

28.3.1.2.20 Interrupt Pin Configuration Register

The interrupt pin configuration register tells which interrupt pin is used (0x01 means INTA). [Figure 28-35](#) shows the interrupt pin configuration register fields.

Offset 0x03D

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	1
W								
Reset	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 28-35. Interrupt Pin

28.3.1.2.21 MIN GNT Configuration Register

Figure 28-36 shows the MIN GNT configuration register fields.

Offset 0x03E

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-36. MIN GNT Configuration Register

28.3.1.2.22 MAX LAT Configuration Register

Figure 28-37 shows the MAX LAT configuration register fields.

Offset 0x03F

Access: User read only

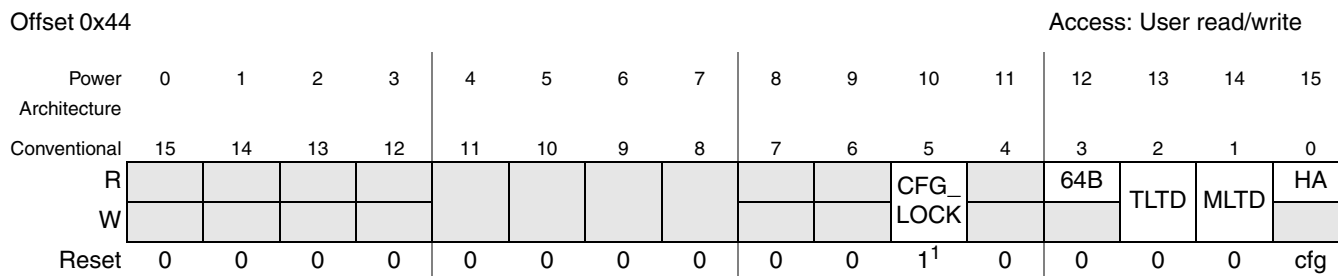
Power Architecture	0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-37. MAX LAT Configuration Register

28.3.1.2.23 PCI Function Configuration Register

Figure 28-38 shows the PCI function configuration register fields.



1. In Agent Mode, contains the value of the pci_aci configuration bit.

= Unimplemented or Reserved

Figure 28-38. PCI Function Configuration Register

Table 28-36. PCI Function Configuration Register Field Descriptions

Field	Description
CFG_LOCK	Configuration Lock. This bit controls access to the configuration spaces of the device from the PCI port. In host mode, the PCI Configuration space is always inaccessible and this bit locks the internal memory-mapped configuration space. Normally, this bit is cleared in agent mode after the configuration of the PCI is complete to allow an external host to configure the rest of the device and access the PCI configuration space. In host mode, this bit would often remain set to prevent other devices from accessing the memory mapped on chip configuration registers accessed through IMMR space. 0 Access to the configuration spaces is permitted. 1 Any inbound PCI access to the PCI configuration space or IMMR space is retried.
TLTD	Target Latency Timeout Disable. This bit determines whether the PCI, while acting as a PCI target, times out when the first data phase of a transaction has not completed in 16 PCI cycles. 0 Target latency timeout enabled. 1 Target latency timeout disabled.
MLTD	Master Latency Timer Disable. This bit determines whether the PCI, while acting as a PCI master, terminates a transaction upon the expiration of the master latency timer. 0 Master latency timer enabled. 1 Master latency timer disabled.
HA	Host/Agent. This bit indicates whether the PCI is in host mode or agent mode. It provides the value of the pci_host—PCI host configuration bit as sampled at the end of the reset sequence. 0 Host Mode 1 Reserved

28.3.1.2.24 PCI Arbiter Control Configuration Register

Figure 28-39 shows the PCI arbiter control configuration register fields.

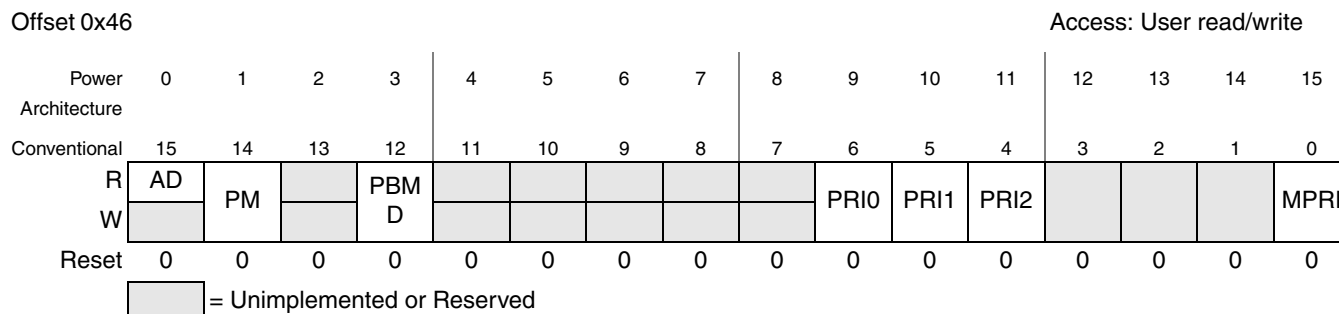


Figure 28-39. PCI Arbiter Control Register

Table 28-37. PCI Arbiter Control Register Field Descriptions

Field	Description
AD	Arbiter disable. This bit indicates whether the PCI functions as the arbiter for the PCI bus. It provides the value of the <code>pci_arb_en</code> — PCI arbiter enable configuration bit as sampled at the end of the reset sequence. 0 Arbiter enabled 1 Arbiter disabled
PM	Parking Mode. This bit controls which device receives a bus grant when there are no outstanding bus requests and the bus is idle. 0 The bus is parked with the last device to use the bus. 1 The bus is parked with the PCI.
PBMD	PCI Broken Master Disable. This bit determines whether the PCI ignores the bus requests of an initiator that requests the bus for an excessive period without using it. 0 An initiator that requests the bus and receives the grant must begin using the bus within 16 PCI clock periods after the bus becomes idle or its request is subsequently ignored. 1 No requests are ignored.
PRI[0:2]	Priority Level for Master <i>N</i> . When the PCI functions as the arbiter for the PCI bus, each PRI _{<i>n</i>} bit determines the arbitration priority level for the PCI master connected to the REQ _{<i>n</i>} /GNT _{<i>n</i>} pair. 0 Low priority 1 High priority
MPRI	My Priority. When the PCI functions as the arbiter for the PCI bus, this bit determines the arbitration priority level for the PCI when it acts as a PCI master. 0 Low priority 1 High priority

28.3.1.3 Software Configuration Registers

Offset 0x48

Access: User read/write

This section describes the software configuration registers that allow a local bus master to access the PCI configuration space, and generate special cycle or interrupt acknowledge transactions on the PCI bus. A special case provides access to the PCI's internal PCI configuration registers.

28.3.1.3.1 CONFIG_ADDRESS

The CONFIG_ADDRESS holds the address for an access to the PCI configuration space from the local bus. This register must be programmed before accessing CONFIG_DATA to perform the transaction. Only 32-bit accesses are permitted.

If EN equals 1, BN equals 0, and DN equals 0, the access is to the internal PCI configuration registers, so no transaction is generated on the PCI bus.

If EN equals 1, BN equals 0, DN equals 31, FN equals 7, and RN equals 0, writing to CONFIG_DATA generates a special cycle transaction and reading from CONFIG_DATA generates an interrupt acknowledge transaction.

Figure 28-37 shows the bit settings of the CONFIG_ADDRESS register.

Table 28-38 shows the CONFIG_ADDRESS register fields.

Offset 0x00																Access: User read/write			
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R	EN								BN										
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	DN				FN				RN										
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
	= Unimplemented or Reserved																		

□ = Unimplemented or Reserved

Figure 28-40. CONFIG_ADDRESS
(Register is repeated for reference.)

Table 28-38. CONFIG_ADDRESS Field Descriptions

Field	Description
EN	Enable Configuration Transaction. This bit determines the type of transaction to be generated. 0 No configuration transaction is generated by accessing the CONFIG_DATA register. Such an access is passed through to the PCI bus as an I/O transaction. Because this is generally not desirable, do not access CONFIG_DATA when the EN bit is 0. 1 A configuration transaction is generated by accessing the CONFIG_DATA register.
BN	Bus Number. This field specifies the bus segment to which a configuration transaction is directed. If this field is 0, a Type 0 configuration transaction is generated. Otherwise, a Type 1 configuration transaction is generated.
DN	Device Number. This field specifies the device to which a configuration transaction is directed. For a Type 0 configuration transaction, this field is decoded to individual IDSEL signals for the address phase according to Table 28-39. For a Type 1 configuration transaction, this field is used directly for the address phase.

Offset 0x00

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN								BN							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DN				FN				RN							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-40. CONFIG_ADDRESS
 (Register is repeated for reference.)

Table 28-38. CONFIG_ADDRESS Field Descriptions (continued)

Field	Description
FN	Function Number. This field specifies the function to which the configuration transaction is directed on a multi-function device. It is used directly in the address phase of the configuration transaction.
RN	Register Number. This field specifies the register being accessed in the PCI configuration space.

Table 28-39. DN Decoding

Value	AD Signal that is Driving High
01010	31
01011	11
01100	12
01101	13
01110	14
01111	15
10000	16
10001	17
10010	18
10011	19
10100	20

Value	AD Signal that is Driving High
10101	21
10110	22
10111	23
11000	24
11001	25
11010	26
11011	27
11100	28
11101	29
11110	30
11111	Special Cycle/Interrupt Acknowledge
Others	Reserved

28.3.1.3.2 CONFIG_DATA

Access to CONFIG_DATA usually generates a PCI configuration transaction if the EN bit of CONFIG_ADDRESS is set. There are some exceptions contained in the description of CONFIG_ADDRESS.

This register may be accessed with a byte, 16-bit, or 32-bit access, depending on the width of the register targeted by the configuration transaction.

Figure 28-41 shows the CONFIG_DATA register fields.

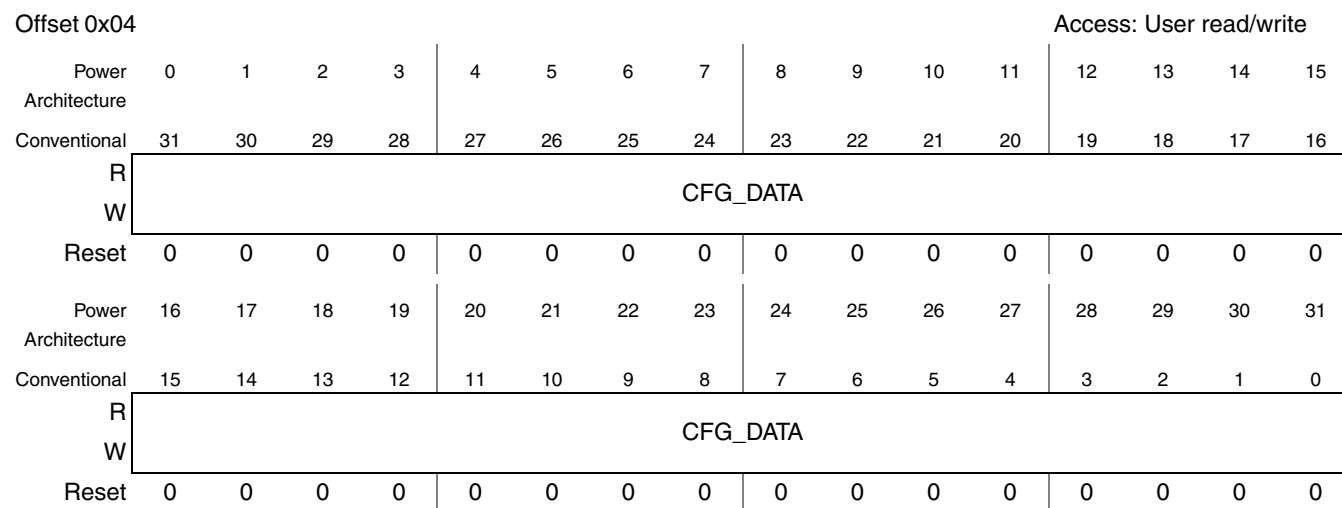


Figure 28-41. CONFIG_DATA

Table 28-40. CONFIG_DATA Field Descriptions

Field	Description
CFG_DATA	Configuration Data. This field contains the data to transferred on a PCI configuration transaction.

28.3.1.3.3 PCI Interrupt Acknowledge Register (PCI_INT_ACK)

Reading this register generates an interrupt acknowledge transaction on the PCI bus. The read value is undefined.

The offset address of this register is 0x08.

28.3.1.4 PCI Bus Arbitration

The PCI bus arbitration approach is access-based. Bus masters must arbitrate for each access performed on the bus. PCI uses a central arbitration scheme where each master has its own unique request (\overline{REQn}) output and grant (\overline{GNTn}) input signal. A simple request-grant handshake is used to gain access to the bus. Arbitration for the bus occurs during the previous access so that no PCI bus cycles are consumed waiting for arbitration (except when the bus is idle).

Three external masters are supported (besides the PCI itself), by using the $\overline{\text{REQ}}$ signals and generating the $\overline{\text{GNT}}$ signals.

During reset, the PCI samples the reset configuration bit (and programs the PCI_ARB_DIS bit accordingly) to determine if the arbiter is enabled or disabled. The arbiter can also be enabled or disabled by directly programming the PCI_ARB_DIS bit in the arbiter configuration register (see [Section 28.3.1.2.24, “PCI Arbiter Control Configuration Register,”](#) for more information).

If the arbiter is disabled, the PCI uses $\overline{\text{REQ0}}$ to issue requests to an external arbiter, and uses $\overline{\text{GNT0}}$ to receive grants from the external arbiter.

28.3.1.4.1 Bus Parking

When no devices are requesting the bus, the bus is granted, or parked, for a specified device to prevent the AD, PCI_C/ $\overline{\text{BE}}$, and PCI_PAR signals from floating. The PCI can be configured to park on itself or park on the last master to use the bus (see [Section 28.3.1.2.24, “PCI Arbiter Control Configuration Register”](#) for more information).

28.3.1.4.2 Arbitration Algorithm

The arbitration algorithm implemented is round-robin with two priority levels. Each of the external PCI bus masters, plus the PCI, are assigned a high or a low priority level, as programmed in the arbiter configuration register (see [Section 28.3.1.2.24, “PCI Arbiter Control Configuration Register,”](#) for more information). Within each priority group (high or low), the bus grant is given to the next requesting device in numerical order, with the PCI itself positioned before device 0. $\overline{\text{GNT}}_n$ is asserted for device n as soon as the previously granted device begins a transaction. Conceptually, the lowest priority device at any given time is the master currently using the bus, and the highest priority device is the next one to follow the current master. This is considered to be a fair algorithm because a given device cannot prevent other devices from having access to the bus—a given device automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, the transaction slot is given to the next requesting device within the priority group.

The grant given to a particular device may be taken away and given to another, higher priority device when the higher priority device asserts its request. If the bus is idle when a new device is to receive a grant, no device receives a grant for one clock. In the next clock, the new winner of the arbitration receives a grant. This operation allows for a turnaround clock when a device is using address stepping or when the bus is parked.

The low priority group collectively receives one bus transaction request slot in the high priority group. Therefore, if there are N high-priority devices, each high-priority device is guaranteed to get at least one of $(N+1)$ bus transactions, and the M low priority devices are guaranteed to each get at least one of $(N+1) \times M$ bus transactions, with one of the low-priority devices receiving the grant in one of $(N+1)$ bus transactions. If all devices are programmed to the same priority level or if there is only one device at the low priority, the algorithm provides each device an equal number of bus grants in a round-robin sequence.

[Figure 28-42](#) shows an arbitration example with three masters in the high priority group and two in the low priority group. With one position in the high priority group as a placeholder for the low priority group, each high priority initiator is guaranteed at least one out of three transaction slots and each low priority initiator is guaranteed at least one out of six slots. Assuming all devices are requesting the bus, the grant

sequence (with device 1 being the current master) is as follows: 0, 2, the PCI, 0, 2, 1, 0, 2, the PCI, and so on. If, for example, device 2 is not requesting the bus, the grant sequence becomes 0, the PCI, 0, 1, 0, the PCI, and so on. If device 2 now requests the bus at a point in the sequence when device 0 is conducting a transaction and the PCI is the next grant, the PCI's grant is removed, and the higher-priority device 2 is awarded the next grant.

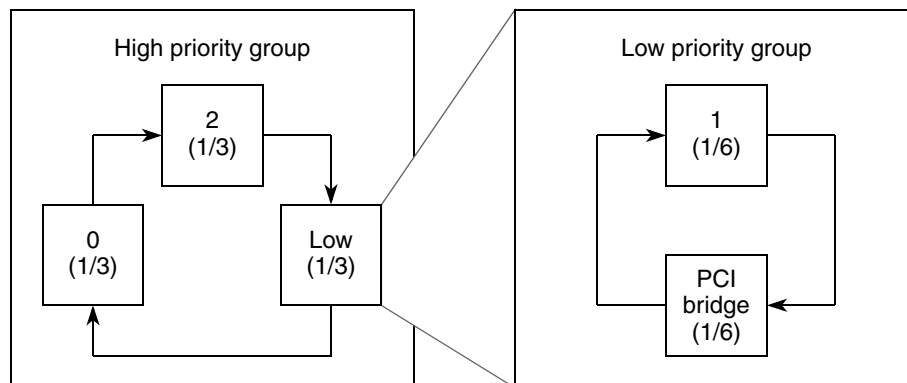


Figure 28-42. PCI Arbitration Example

28.3.1.4.3 Broken Master Lock-Out

The PCI bus arbiter on the PCI has a feature that allows it to lock out any broken or ill-behaved masters. The broken master feature is controlled by programming the PCI arbiter control register. When the broken master feature is enabled, a granted device that does not assert `PCI_FRAME` within 16 PCI clock cycles after the bus is idle has its grant removed and subsequent requests are ignored until its `REQ` is negated for at least one clock cycle. This prevents ill-behaved masters from monopolizing the bus. When the broken master feature is disabled, a device that requests the bus and receives a grant never loses its grant until it begins a transaction or negates its `REQ` signal. Disabling the broken master feature is not recommended.

28.3.1.4.4 Master Latency Timer

The PCI implements the master latency timer register (see [Section 28.3.1.2.10, “Latency Timer Configuration Register,”](#) for more information) to prevent itself from monopolizing the bus. When the master latency timer expires, the PCI checks the state of its $\overline{\text{GNT}}$ signals. If the $\overline{\text{GNT}}$ signal is not asserted, the PCI completes one more data phase and relinquishes the bus. The master latency timer can be disabled if needed (see [Section 28.3.1.2.23, “PCI Function Configuration Register,”](#) for more information).

28.4 PCI Interface Functional Description

The following sections discuss the operation of the PCI bus.

28.4.1 Bus Commands

PCI bus commands indicate the type of transaction occurring on the bus. These commands are encoded on $\text{PCI_C/BE}[3-0]$ during the address phase of the transaction. PCI bus commands are described in [Table 28-41](#).

Table 28-41. PCI Command Definitions

PCI_C/ BE[3–0]	Command Type	Supported as:		Definition
		Initiator	Target	
0b0000	Interrupt acknowledge	YES	NO	A read implicitly addressed to the system interrupt controller. The size of the vector to be returned is indicated on the byte enables after the address phase.
0b0001	Special cycle	YES	NO	Provides a simple message broadcast mechanism. See Section 28.4.1.2.5, “Special Cycle Command,” for more information.
0b0010	I/O read	YES	NO	Accesses agents mapped in I/O address space.
0b0011	I/O write	YES	NO	Accesses agents mapped in I/O address space.
0b010x	—	—	—	Reserved. No response occurs.
0b0110	Memory read	YES	YES	Accesses agents mapped in memory address space. A read from prefetchable space, when seen as a target, fetches a cache line of data (32 bytes) from the starting address, even though all 32 bytes may not actually be sent to the initiator.
0b0111	Memory write	YES	YES	Accesses agents mapped in memory address space.
0b100x	—	—	—	Reserved. No response occurs.
0b1010	Configuration read	YES	YES	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 28.4.1.2.4, “Host Mode Configuration Access,” for more information on configuration accesses. As a target, a configuration read is only accepted if the PCI is configured to be in agent mode.

Table 28-41. PCI Command Definitions (continued)

PCI_C/ BE[3–0]	Command Type	Supported as:		Definition
		Initiator	Target	
0b1011	Configuration write	YES	YES	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 28.4.1.2.4, “Host Mode Configuration Access,” for more information. As a target, a configuration write is only accepted if the PCI is configured to be in agent mode.
0b1100	Memory read multiple	YES	YES	Causes a prefetch of the next cache line.
0b1101	Dual address cycle	NO	YES	Transfers an 8-byte address to devices.
0b1110	Memory read line	YES	YES	Indicates that the initiator intends to transfer an entire cache line of data.
0b1111	Memory write and invalidate	NO	YES	Indicates that the initiator transfers an entire cache line of data, and if PCI has any cacheable memory, this line needs to be invalidated.

28.4.1.1 PCI Protocol Fundamentals

The bus transfer mechanism on the PCI bus is called a burst. A burst is comprised of an address phase and one or more data phases.

All signals are sampled on the rising edge of the PCI clock. Each signal has a setup and hold window with respect to the rising clock edge, in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance.

28.4.1.1.1 Basic Transfer Control

PCI data transfers are controlled by the following three fundamental signals:

- $\overline{\text{PCI_FRAME}}$ is driven by an initiator to indicate the beginning and end of a transaction.
- $\overline{\text{PCI_IRDY}}$ (initiator ready) is driven by an initiator, allowing it to force wait cycles.
- $\overline{\text{PCI_TRDY}}$ (target ready) is driven by a target, allowing it to force wait cycles.

The bus is idle when $\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ are negated. The first clock cycle in which $\overline{\text{PCI_FRAME}}$ is asserted indicates the beginning of the address phase. The address and the bus command code are transferred in that cycle. The next cycle ends the address phase and begins the data phase.

During the data phase, data is transferred in each cycle that $\overline{\text{PCI_IRDY}}$ and $\overline{\text{PCI_TRDY}}$ are asserted. After the PCI, as an initiator, has asserted $\overline{\text{PCI_IRDY}}$, it does not change $\overline{\text{PCI_IRDY}}$ or $\overline{\text{PCI_FRAME}}$ until the current data phase completes, regardless of the state of $\overline{\text{PCI_TRDY}}$. After the PCI, as a target, has asserted $\overline{\text{PCI_TRDY}}$ or $\overline{\text{PCI_STOP}}$, it does not change $\overline{\text{PCI_DEVSEL}}$, $\overline{\text{PCI_TRDY}}$, or $\overline{\text{PCI_STOP}}$ until the current data phase completes.

When the PCI (as a master) intends to complete only one more data transfer, $\overline{\text{PCI_FRAME}}$ is negated and $\overline{\text{PCI_IRDY}}$ is asserted (or kept asserted) indicating the initiator is ready. After the target indicates it is ready ($\overline{\text{PCI_TRDY}}$ asserted) the bus returns to the idle state.

28.4.1.1.2 Addressing

The PCI specification defines three physical address spaces—memory, I/O, and configuration. The memory and I/O address spaces are standard for all systems. The configuration address space has been defined specifically to support PCI hardware configuration. Each PCI device decodes the address for each PCI transaction with each agent responsible for its own address decode.

The information contained in the two lower address bits (AD1 and AD0) depends on the address space. In the I/O address space, all 32 address/data lines provide the full byte address. AD[1–0] are used for the generation of $\overline{\text{PCI_DEVSEL}}$ and indicate the least significant valid byte involved in the transfer. After a target has claimed an I/O access, it first determines if it can complete the entire access as indicated by the byte enable signals. If all the selected bytes are not in the address range, the entire access should not be completed; the target should not transfer any data and should terminate the transaction with a target-abort. See [Section 28.4.1.1.6, “Bus Transactions,”](#) for more information.

In the configuration address space, accesses are decoded to a 4-byte address using AD[7–2]. An agent determines if it is the target of the access when a configuration command is decoded, IDSEL is asserted, and AD[1–0] are 0b00; otherwise, the agent ignores the current transaction. The PCI determines a configuration access is for a device on the PCI bus by decoding a configuration command. When in agent mode, the PCI responds to host-generated PCI configuration cycles when its IDSEL is asserted during a configuration cycle.

For memory accesses, the address is decoded using AD[31–2]; thereafter, the address is incremented internally by four bytes until the end of the burst transfer. Another initiator in a memory access should drive 0b00 on AD[1–0] during the address phase to indicate a linear incrementing burst order. The PCI checks AD[1–0] during a memory command access and provides the linear incrementing burst order. On reads, if AD[1–0] is 0b10, which represents a cache line wrap, the PCI linearly increments the burst order starting at the critical 64-bit address, wraps at the end of the cache line, and disconnects after reading one cache line. If AD[1–0] is 0bx1 (a reserved encoding) and the PCI_C/ $\overline{\text{BE}}$ [3–0] signals indicate a memory transaction, it executes a target disconnect after the first data phase is completed. AD[1–0] are included in parity calculations.

28.4.1.1.3 Device Selection

As a target, the PCI drives $\overline{\text{PCI_DEVSEL}}$ one clock following the address phase as indicated in the configuration space status register; see [Section 28.3.1.2.4, “PCI Status Configuration Register,”](#) for more information. The PCI as a target qualifies the address/data lines with $\overline{\text{PCI_FRAME}}$ before asserting $\overline{\text{PCI_DEVSEL}}$. The $\overline{\text{PCI_DEVSEL}}$ signal is asserted at or before the clock edge at which the PCI enables its $\overline{\text{PCI_TRDY}}$, $\overline{\text{PCI_STOP}}$, or data (for a read). The $\overline{\text{PCI_DEVSEL}}$ signal is not negated until $\overline{\text{PCI_FRAME}}$ is negated, with $\overline{\text{PCI_IRDY}}$ asserted and either $\overline{\text{PCI_STOP}}$ or $\overline{\text{PCI_TRDY}}$ asserted. The exception to this is a target-abort; see [Section 28.4.1.1.8, “Transaction Termination,”](#) for more information.

As an initiator, if the PCI does not see the assertion of $\overline{\text{PCI_DEVSEL}}$ within four clocks of $\overline{\text{PCI_FRAME}}$, it terminates the transaction with a master-abort as described in [Section 28.4.1.1.8, “Transaction Termination,”](#) for more information.

28.4.1.1.4 Byte Enable Signals

The byte enable signals ($\overline{\text{BE}}[3-0]$) indicate which byte lanes carry valid data. The byte enable signals may enable different bytes for each of the data phases. The byte enable signals are valid on the edge of the clock that starts each data phase and remain valid for the entire data phase.

If the PCI, as a target, sees no byte enable signals asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the PCI expects the data not to be changed, and on a write transaction, the data is not stored.

28.4.1.1.5 Bus Driving and Turnaround


The turnaround-cycle is one clock cycle and is required to avoid contention. This cycle occurs at different times for different signals. $\overline{\text{PCI_IRDY}}$, $\overline{\text{PCI_TRDY}}$, and $\overline{\text{PCI_DEVSEL}}$ use the address phase as their turnaround-cycle. $\overline{\text{PCI_FRAME}}$, $\overline{\text{PCI_C/BE}}[3-0]$, and $\text{AD}[31-0]$ use the idle cycle between transactions as their turnaround-cycle. (An idle cycle in PCI is when both $\overline{\text{PCI_FRAME}}$ and $\overline{\text{PCI_IRDY}}$ are negated.)

Byte lanes not involved in the current data transfer are driven to a stable condition even though the data is not valid.

28.4.1.1.6 Bus Transactions

The timing diagrams in this section show the relationship of significant signals involved in bus transactions.

The following conventions are important.

- When a signal is drawn as a solid line, it is actively being driven by the current initiator or target.
- When a signal is drawn as a dashed line, no agent is actively driving it.
- Three-stated signals with slashes between the two rails have indeterminate values.
- The terms edge and clock edge refer to the rising edge of the clock.
- The terms asserted and negated refer to the globally visible state of the signal on the clock edge, and not to signal transitions.
- The symbol  represents a turnaround-cycle.

28.4.1.1.7 Read and Write Transactions

Read and write transactions begin with an address phase followed by a data phase. The address phase occurs when $\overline{\text{PCI_FRAME}}$ is asserted for the first time, the $\text{AD}[31-0]$ signals contain a byte address, and the $\overline{\text{PCI_C/BE}}[3-0]$ signals contain a bus command. The data phase consists of the actual data transfer and possible wait cycles; the byte enable signals remain actively driven from the first clock of the data phase through the end of the data transfer.

A read transaction starts when $\overline{\text{PCI_FRAME}}$ is asserted for the first time and the $\text{PCI_C}/\overline{\text{BE}}[3-0]$ signals indicate a read command. Figure 28-43 shows an example of a single beat read transaction.

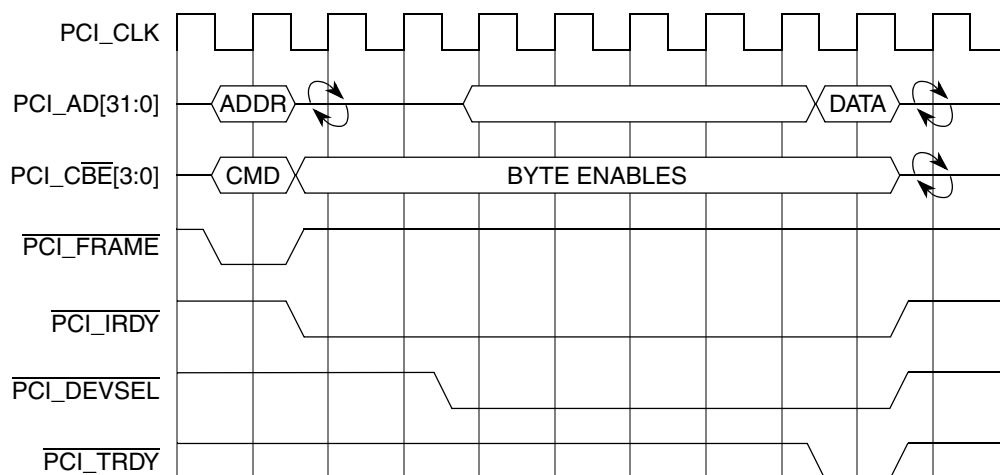


Figure 28-43. Single Beat Read Example

Figure 28-44 shows an example of a burst read transaction.

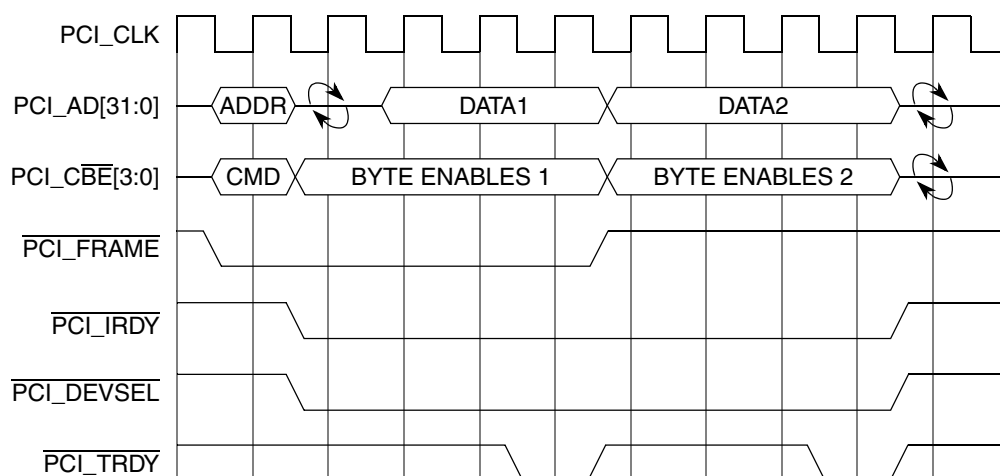


Figure 28-44. Burst Read Example

During the turnaround-cycle following the address phase, the $\text{PCI_C}/\overline{\text{BE}}[3-0]$ signals indicate which byte lanes are involved in the data phase. The turnaround-cycle must be enforced by the target with the PCI_TRDY signal if using fast PCI_DEVSEL assertion. The earliest the target can provide valid data is one cycle after the turnaround-cycle. The target must drive the $\text{AD}[31-0]$ signals when PCI_DEVSEL is asserted except during the turnaround-cycle.

The data phase completes when data is transferred, which occurs when PCI_IRDY and PCI_TRDY are asserted on the same clock edge. When either is negated a wait cycle is inserted and no data is transferred. To indicate the last data phase, PCI_IRDY must be asserted when PCI_FRAME is negated.

A write transaction starts when $\overline{\text{PCI_FRAME}}$ is asserted for the first time and the $\text{PCI_C}/\overline{\text{BE}}[3:0]$ signals indicate a write command. Figure 28-45 shows an example of a single beat write transaction.

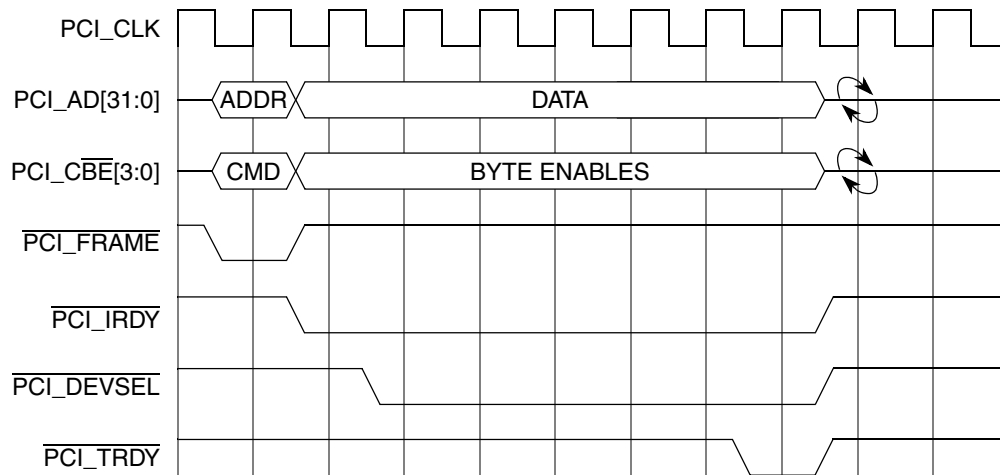


Figure 28-45. Single Beat Write Example

Figure 28-46 shows an example of a burst write transaction.

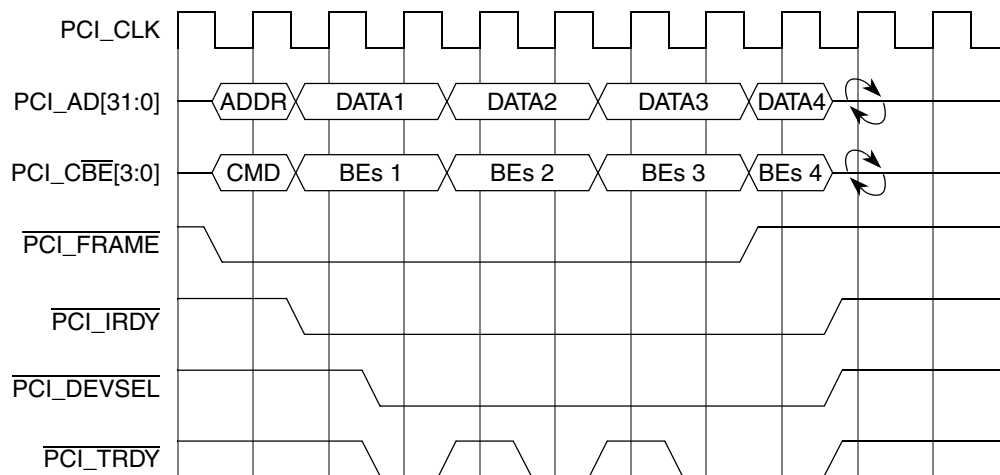


Figure 28-46. Burst Write Example

A write transaction is similar to a read transaction except no turnaround cycle is needed following the address phase because the initiator provides address and data. Data phases are the same for read and write transactions.

28.4.1.1.8 Transaction Termination

The termination of a PCI transaction is orderly and systematic, regardless of the cause of the termination. All transactions end when $\overline{\text{PCI_FRAME}}$ and PCI_IRDY are both negated, indicating the idle cycle.

The PCI as an initiator terminates a transaction when $\overline{\text{PCI_FRAME}}$ is negated and PCI_IRDY is asserted. This indicates the final data phase is in progress. The final data transfer occurs when PCI_TRDY and PCI_IRDY are asserted. A master-abort is an abnormal case of a master initiated termination. If the PCI

detects that $\overline{\text{PCI_DEVSEL}}$ has remained negated for more than four clocks after the assertion of $\overline{\text{PCI_FRAME}}$, it negates $\overline{\text{PCI_FRAME}}$ and then, on the next clock, negates $\overline{\text{PCI_IRDY}}$. On aborted reads, the PCI returns 0xFFFF_FFFF. The data is lost on aborted writes.

When the PCI as a target needs to suspend a transaction, it asserts $\overline{\text{PCI_STOP}}$. After asserted, $\overline{\text{PCI_STOP}}$ remains asserted until $\overline{\text{PCI_FRAME}}$ is negated. Depending on the circumstances, data may or may not be transferred during the request for termination. If $\overline{\text{PCI_TRDY}}$ and $\overline{\text{PCI_IRDY}}$ are asserted during the assertion of $\overline{\text{PCI_STOP}}$, data is transferred. This type of target-initiated termination is called a disconnect B, shown in Figure 28-47. If $\overline{\text{PCI_TRDY}}$ is asserted when $\overline{\text{PCI_STOP}}$ is asserted but $\overline{\text{PCI_IRDY}}$ is not, $\overline{\text{PCI_TRDY}}$ must remain asserted until $\overline{\text{PCI_IRDY}}$ is asserted and the data is transferred. This is called a disconnect A target-initiated termination, also shown in Figure 28-47. However, if $\overline{\text{PCI_TRDY}}$ is negated when $\overline{\text{PCI_STOP}}$ is asserted, no more data is transferred. Therefore, the initiator does not have to wait for a final data transfer (see the retry diagram in Figure 28-47).

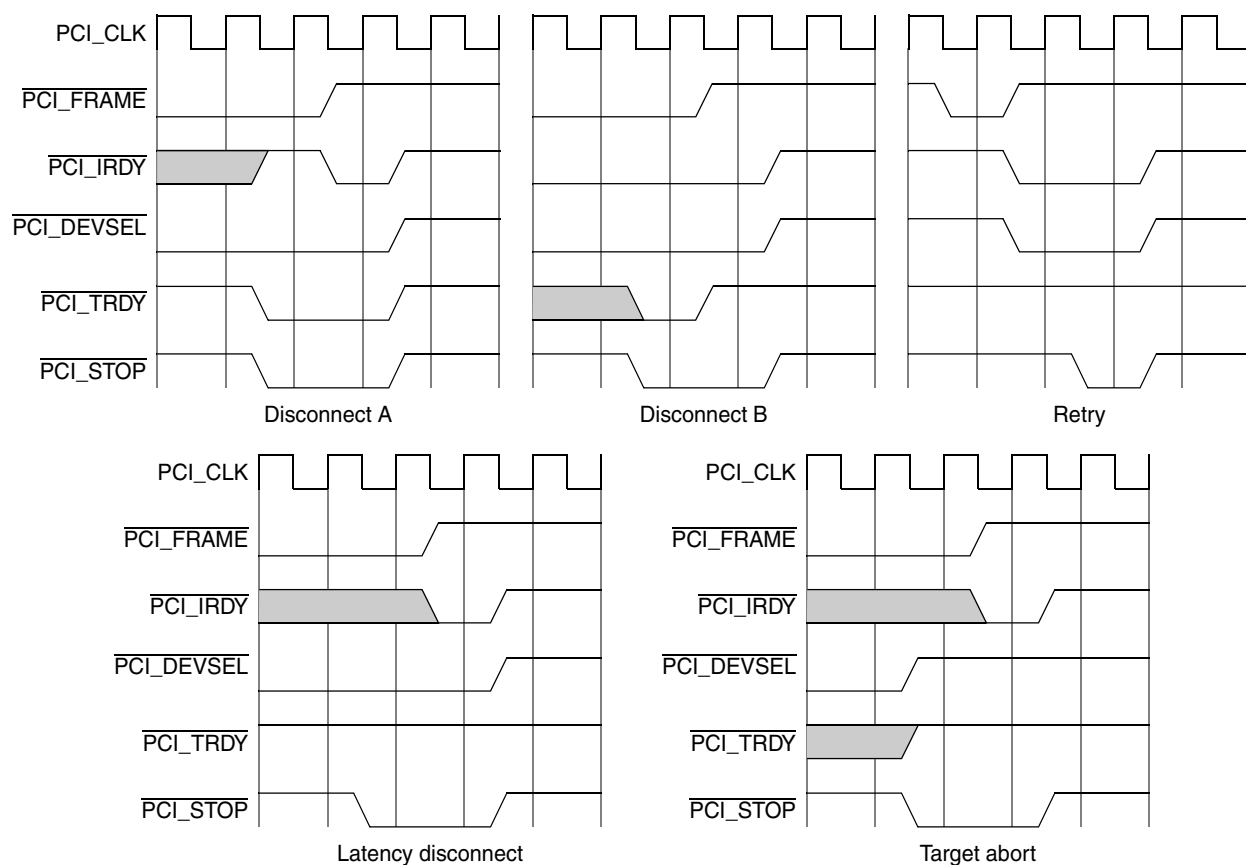


Figure 28-47. Target-Initiated Terminations

When an initiator is terminated by $\overline{\text{PCI_STOP}}$, it must negate its $\overline{\text{REQX}}$ signal for a minimum of two PCI clocks (of which one clock is needed for the bus to return to the idle state). If the initiator intends to complete the transaction, it should reassert its $\overline{\text{REQX}}$ immediately following the two clocks or potential starvation may occur. If the initiator does not intend to complete the transaction, it can assert $\overline{\text{REQX}}$ when it needs to use the PCI bus again.

The PCI terminates a transaction if:

- Eight PCI clock cycles have elapsed between data phases. This is a latency disconnect (see [Figure 28-47](#)).
- AD[1–0] is 0bx1 (a reserved burst ordering encoding) during the address phase and one data phase has completed.
- The PCI command is a configuration command and one data phase has completed.
- A streaming transaction crosses a 4K page boundary.
- A streaming transaction runs out of I/O sequencer buffer entries.
- A cache line wrap transaction has completed a cache line transfer.

Another target-initiated termination is the retry termination. Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. This can occur because no buffer entries are available in the I/O sequencer, or the sixteen clock latency timer has expired without transfer of the first data. The target latency timer of the PCI can be optionally disabled see [Section 28.3.1.2.23, “PCI Function Configuration Register.”](#)

When the PCI is in host mode, it does not respond to any PCI configuration transactions. Because a target can determine whether or not data is transferred (when both $\overline{\text{PCI_IRDY}}$ and $\overline{\text{PCI_TRDY}}$ are asserted), if it wants to do only one more data transfer and then stop, it may assert $\overline{\text{PCI_TRDY}}$ and $\overline{\text{PCI_STOP}}$ at the same time.

Target-abort refers to the abnormal termination used when a fatal error has occurred or when a target can never respond. Target-abort is indicated when $\overline{\text{PCI_STOP}}$ is asserted and $\overline{\text{PCI_DEVSEL}}$ is negated. This indicates that the target requires the transaction to be terminated and does not want the transaction tried again. Any transferred data may have been corrupted.

The PCI terminates a transaction with target-abort if it is the intended target of a read transaction from system memory and the data from memory is corrupt. If the PCI is the intended target of a transaction and an address parity error occurs, or a data parity error occurs on a write transaction to system memory, it continues the transaction on the PCI bus but aborts internally. The PCI does not target-abort in this case.

If the PCI is mastering a transaction and the transaction terminates with a target-abort, undefined data is returned on a read and write data is lost. [Figure 28-47](#) shows an example of a target-abort.

An initiator may retry any target disconnect accesses, except target-abort, at a later time starting with the address of the next non-transferred data. Retry is actually a special case of disconnect where no data transfer occurs at all and the initiator must start the entire transaction over again.

28.4.1.2 Other Bus Operations

The following sections provide information on additional PCI bus operations.

28.4.1.2.1 Fast Back-to-Back Transactions

In the two types of fast back-to-back transactions, the first type places the burden of avoiding contention on the initiator while the second places the burden on all potential targets. The PCI as a target supports

both types of fast back-to-back transactions, but does not support them as an initiator. The PCI as a target has the fast back-to-back enable bit hardwired to one (enabled).

For the first type (governed by the initiator), the initiator may only run a fast back-to-back transaction to the same target. For the second type, when the PCI detects a fast-back-to-back operation and did not drive `PCI_DEVSEL` in the previous cycle, it delays the assertion of `PCI_DEVSEL` and `PCI_TRDY` for one cycle to allow the other target to get off the bus.

28.4.1.2.2 Dual Address Cycles

The PCI supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as a target (outbound DAC is not supported). DACs are different from single address cycles (SACs) in that the address phase takes two PCI beats instead of one PCI beat to transfer (64-bit vs. 32-bit addressing). Only PCI memory commands can use DAC cycles; I/O, configuration, interrupt acknowledge, and special cycle command cannot use DAC cycles. The PCI supports single-beat and burst DAC transactions.

28.4.1.2.3 Data Streaming

The PCI provides data streaming for PCI transactions to and from prefetchable memory. In other words, when the PCI is a target for a PCI initiated transaction, it supplies or accepts multiple cache lines of data without disconnecting. For PCI transactions to non-prefetchable space, the PCI disconnects after the first data phase so that no streaming can occur.

For PCI memory reads, streaming is achieved by performing speculative reads from memory in prefetchable space. A block of memory may be marked as prefetchable by setting the PCI configuration registers bit for the inbound address translation (see [Section 28.3.1.1.14, “PCI Inbound Window Attribute Registers \(PIWARn\),”](#) for more information) when:

- Reads do not alter the contents of memory (reads have no side effects)
- Reads return all bytes regardless of the byte enable signals
- Writes can be merged without causing errors

For a memory read command or a memory read line command, the PCI reads one cache line from memory. If the PCI read or read line transaction crosses a cache line boundary, the PCI starts the read of a new cache line. For a memory read multiple command, the PCI reads two cache lines from memory. When the PCI transaction finishes the read for the first cache line, the PCI performs a speculative read of a third cache line. The PCI continues this prefetching until the end of the transaction.

For PCI writes to memory, streaming is achieved by buffering the transaction in the space available within the I/O sequencer. This allows PCI memory writes to execute with no wait states.

A disconnect occurs if the PCI runs out of buffer space on writes, or the PCI cannot supply consecutive data beats for reads within eight PCI bus clocks of each other. A disconnect also occurs if the transaction crosses a 4K page boundary.

28.4.1.2.4 Host Mode Configuration Access

The PCI provides two types of configuration accesses to support hierarchical bridges. To access configuration space, a value is written to the CONFIG_ADDR register specifying which PCI bus, which device, and which configuration register to be accessed.

When the PCI sees an access that falls inside the four bytes beginning at the CONFIG_DATA address, it checks the enable bit, the device number, and the bus number in the CONFIG_ADDR register. If the enable bit is set and the device number is not equal to all ones, a configuration cycle translation is performed. When the device number field is equal to all ones, it has a special meaning (see [Section 28.4.1.2.5, “Special Cycle Command,”](#) for more information).

There are two types of translations supported:

- Type 0 translations—For when the device is on the PCI bus connected to the PCI.
- Type 1 translations—For when the device is on another bus somewhere behind the PCI.

For Type 0 translations, the PCI decodes the device number field to assert the appropriate IDSEL line and perform a configuration cycle on the PCI bus with AD[1-0] as 0b00. All 21 IDSEL bits are decoded, starting with bit AD[11]. If the device number field contains 0b01011, AD[11] on the PCI bus is set. The IDSEL lines are bit-wise associated with increasing values for the device number such that AD[12] corresponds to 0b01100, and so on up to bit 30 as shown in [Table 28-40](#). AD[31] is selected with 0b01010. A device number of 0b11111 indicates a special cycle. Device number 0b00000 is used for configuring the PCI itself. Bits 10 through 8 are copied to the PCI bus as an encoded value for components that contain multiple functions. Bits 7 through 2 are also copied onto the PCI bus. The PCI implements address stepping on configuration cycles so the target’s IDSEL, which is connected directly to one of the AD lines, reaches a stable value. This means that a valid address and command are driven on the AD and PCI_C/BE lines one cycle before the assertion of PCI_FRAME.

For Type 1 translations, the PCI copies the contents of the CONFIG_ADDR register directly onto the PCI address/data lines during the address phase of a configuration cycle, with the exception that AD[1-0] contains 0b01 (not 0b00 as in Type 0 translations).

When the PCI is configured as a host device, a local master sometimes needs to perform configuration reads from unpopulated PCI slots (as part of the system configuration). To avoid getting a machine check interrupt, the following steps should be taken:

1. Mask the NORSP bit in the error mask register. See [Section 28.3.1.1.1, “PCI Error Status Register \(PCI_ESR\),”](#) for more information.
2. Perform the PCI configuration reads.
3. Clear the NORSP bit in the error status register.
4. Unmask (write '1') the NORSP bit in the error mask register. See [Section 28.3.1.1.3, “PCI Error Enable Register \(PCI_EER\),”](#) for more information.

28.4.1.2.5 Special Cycle Command

A special cycle command contains no explicit destination address, but is broadcast to all PCI agents. Each receiving agent must determine whether the message is applicable to itself. No assertion of PCI_DEVSEL in response to a special cycle command is necessary.

A special cycle command is like any other bus command because it has an address phase and a data phase. The address phase starts like all other commands with the assertion of `PCI_FRAME` and completes when `PCI_FRAME` and `PCI_IRDY` are negated. Special cycles terminate with a master-abort. In the special cycle case, the received-master-abort bit in the configuration status register is not set.

The address phase contains no valid information other than the command field. Even though there is no explicit address, the address/data lines are driven to a stable state and parity is generated. During the data phase, the address/data lines contain the message type and an optional data field. The message is encoded on the sixteen least-significant bits (`AD[15-0]`). The data field is encoded on `AD[31-16]`. When running a special cycle, the message and data are valid on the first clock `PCI_IRDY` is asserted.

When the `CONFIG_ADDRESS` register gets written with a value so the bus number matches the bridge's bus, the device number is all ones, the function number is all ones and the register number is zero. The next time the `CONFIG_DATA` register is accessed the PCI does a special cycle or an interrupt acknowledge command. When the `CONFIG_DATA` register is written, the PCI generates a special cycle encoding on the command/byte enable lines during the address phase, and drives the data from the `CONFIG_DATA` register onto the address/data lines during the first data phase.

If the bus number field of the `CONFIG_ADDRESS` does not match one of the PCI's bus numbers, the PCI passes the write to `CONFIG_DATA` on through to the PCI bus as a type 1 configuration cycle.

28.4.1.2.6 Interrupt Acknowledge

When the `CONFIG_ADDRESS` register gets written with a value so the bus number is 0x00, the device number is all ones, the function number is all ones and the register number is zero. The next time the `CONFIG_DATA` register is accessed, the PCI does a special cycle command or an interrupt acknowledge command. When the `CONFIG_DATA` register is read, the PCI generates an interrupt acknowledge command encoding on the command/byte enable lines during the address phase. During the address phase, `AD[31-0]` does not contain a valid address, but are driven with stable data and valid parity (`PCI_PAR`). During the data phase, the byte enable signals determine which bytes are involved in the transaction. The interrupt vector must be returned when `PCI_TRDY` is asserted.

An interrupt acknowledge transaction can also be issued on the PCI bus by reading from the `PCI_INT_ACK` register.

28.4.1.2.7 Error Functions

This section discusses PCI bus errors.

28.4.1.2.8 Parity

During valid 32-bit address and data transfers, parity covers all 32 address/data lines and the four command/byte enable lines regardless of whether or not all lines carry meaningful information. Byte lanes not actually transferring data are driven with stable (albeit meaningless) data and included in the parity calculation. During configuration, special cycle, or interrupt acknowledge commands, some address lines are not defined, but remain driven to stable values and included in the parity calculation.

Even parity is calculated for all PCI operations. The value of `PCI_PAR` is generated so the number of ones on `PCI_AD[31:0]`, `PCI_CBE[3:0]`, and `PCI_PAR` equals an even number. The `PCI_PAR` signal is driven when the address/data lines are driven and follow the corresponding address or data by one clock.

The PCI checks the parity after all valid address phases (the assertion of `PCI_FRAME`) and for valid data transfers (`PCI_IRDY` and `PCI_TRDY` asserted) involving the PCI. When an address or data parity error is detected, the detected-parity-error bit in the configuration space status register is set (see [Section 28.3.1.2.4, “PCI Status Configuration Register,”](#) for more information).

28.4.1.2.9 Error Reporting

Except for setting the detected-parity-error bit, all parity error reporting and response is controlled by the parity-error-response bit (see [Section 28.3.1.2.3, “PCI Command Configuration Register,”](#) for more information). If the parity-error-response bit is cleared, the PCI completes all transactions regardless of parity errors (address or data). If the bit is set, the PCI asserts `PCI_PERR` two clocks after the actual data transfer in which a data parity error is detected and keeps `PCI_PERR` asserted for one clock. The PCI asserts `PCI_PERR` when acting as an initiator during a read transaction or as a target involved in a write to system memory. [Figure 28-48](#) shows the possible assertion points for `PCI_PERR` if the PCI detects a data parity error.

As an initiator, the PCI attempts to complete the transaction on the PCI bus if a data parity error is detected and sets the data-parity-reported bit in the configuration space status register. If a data parity error occurs on a read transaction, the PCI aborts the transaction internally. As a target, the PCI completes the transaction on the PCI bus even if a data parity error occurs. If parity error occurs during a write to system memory, the transaction completes on the PCI bus but is aborted internally, ensuring that potentially corrupt data does not go to memory.

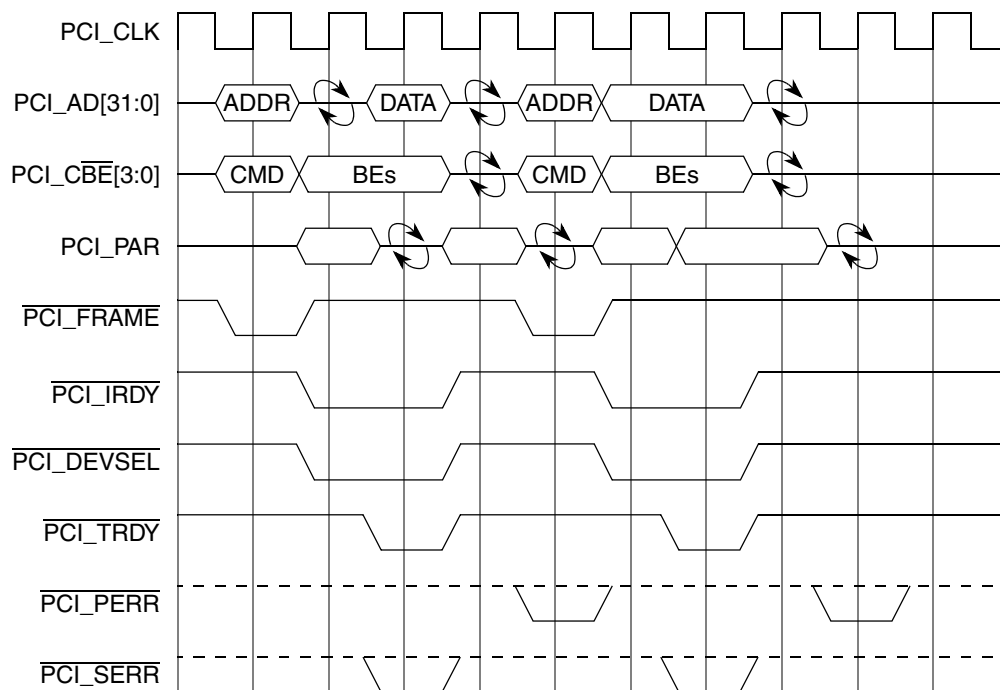


Figure 28-48. PCI Parity Operation

When the PCI asserts $\overline{\text{PCI_SERR}}$, it sets the signaled-system-error bit in the configuration space status register. Additionally, if the error is an address parity error, the parity-error-detected bit is set; reporting an address parity error on $\overline{\text{PCI_SERR}}$ is conditioned on the parity-error-response bit being enabled in the command register. $\overline{\text{PCI_SERR}}$ is asserted when the PCI detects an address parity error while acting as a target. The system error is passed to the PCI's interrupt processing logic to assert $\overline{\text{MCP}}$. [Figure 28-48](#) shows where the PCI could detect an address parity error and assert $\overline{\text{PCI_SERR}}$ or where the PCI, acting as an initiator, checks for the assertion of $\overline{\text{PCI_SERR}}$ signaled by the target detecting an address parity error.

As a target that asserts $\overline{\text{PCI_SERR}}$ on an address parity, the PCI completes the transaction on the PCI bus, aborting internally if the transaction is a write to system memory. If $\overline{\text{PCI_PERR}}$ is asserted during a PCI write to PCI, the PCI attempts to continue the transfer, allowing the target to abort/disconnect if desired. If the PCI detects a parity error on a read from PCI, the PCI aborts the transaction internally and continues the transfer on the PCI bus, allowing the target to abort/disconnect if desired.

In all cases of parity errors on the PCI bus, regardless of the parity-error-response bit, information about the transaction is logged in the PCI error control capture register, the PCI error address capture register, and the PCI error data capture register. $\overline{\text{MCP}}$ is also asserted to the core as an option.

28.4.1.3 PCI Inbound Address Translation

For inbound transactions (transactions generated by an external master on the PCI bus where the PCI responds as a slave device), the PCI only responds to PCI addresses within the windows mapped by the PCI inbound base address registers (PIBARs). If there is an address hit in one of the PIBARs, the PCI address is translated from PCI space to local memory space through the associated PCI inbound translation address registers (PITARs). This allows an external master to access local memory. Each PIBAR register is associated with a PITAR and PIWAR located in the PCI's PCI CSR space. [Figure 28-49](#) shows an example translation window for inbound memory accesses.

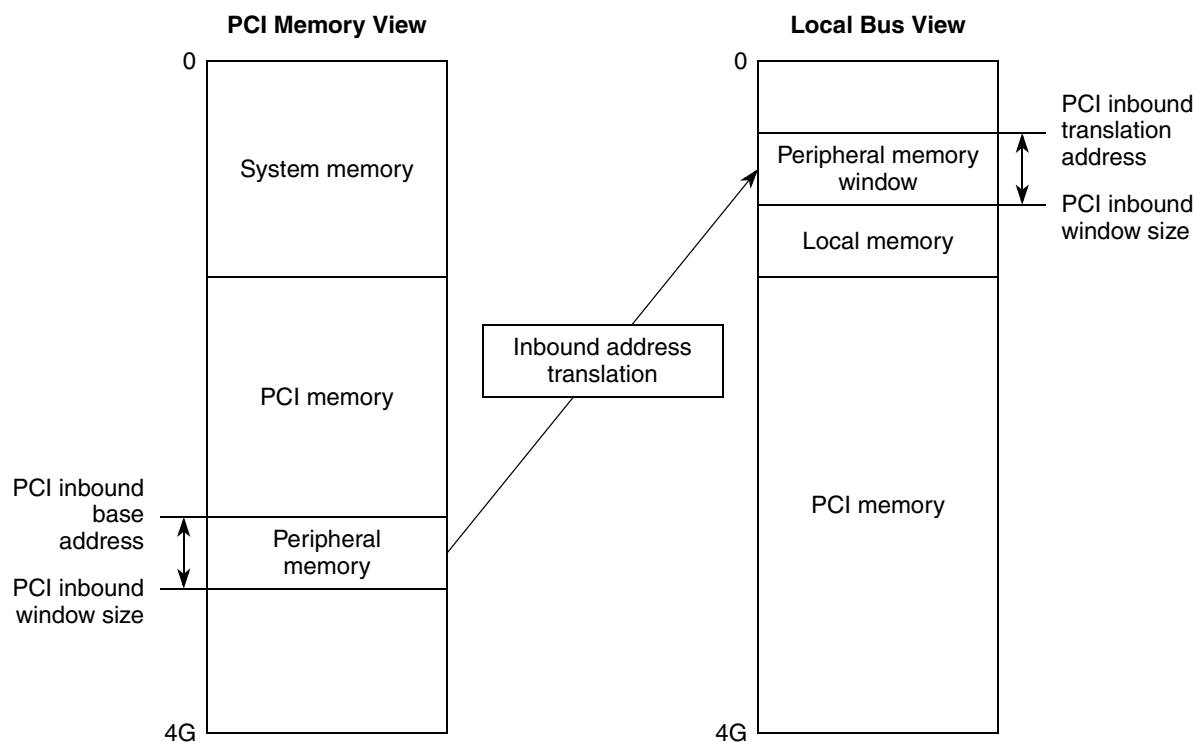


Figure 28-49. Inbound PCI Memory Address Translation

There are three full sets of inbound translation registers, in addition to the PIMMR base address register, allowing four simultaneous translation windows (one to a fixed destination and three programmable). Only two of the programmable windows can be mapped anywhere in the 64-bit PCI address space. Window 0 can only be mapped within the lowest 4-Gbyte space. Software can move the programmable translation base addresses during run-time to access different portions of local memory. PCI inbound translation windows may not overlap.

The translation windows are disabled after reset. PCI does not acknowledge externally mastered transactions on the PCI bus by asserting `PCI_DEVSEL` until the inbound translation windows are enabled.

28.5 I/O Sequencer for PCI Subsystem (PCI)

28.6 Introduction

The PCI_IOS switches transactions among its ports, using a buffer pool to minimize blocking. Figure 28-49 is a block diagram of the I/O sequencer (PCI_IOS).

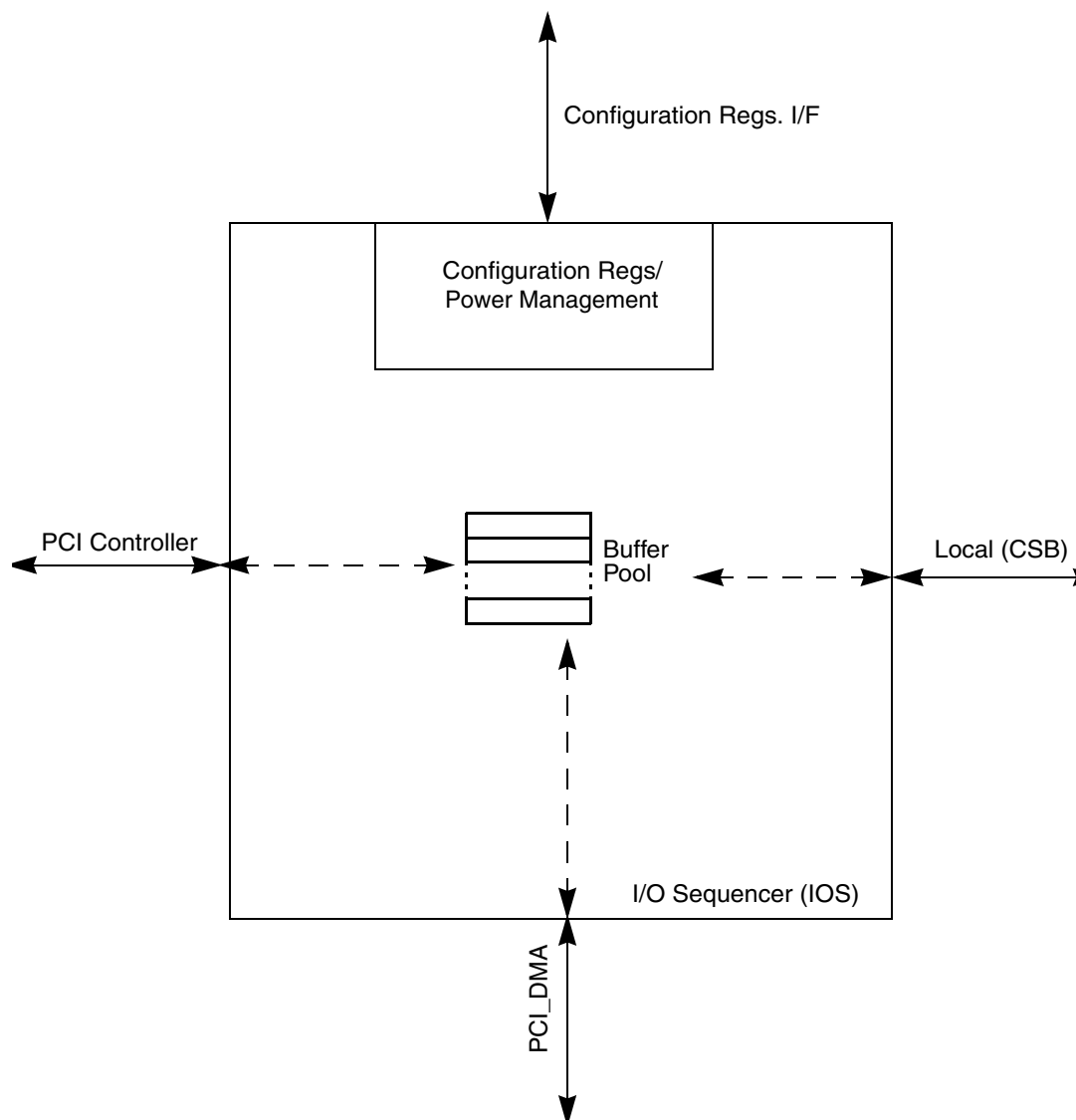


Figure 28-50. PCI_IOS Block Diagram

28.6.1 Features

The PCI_IOS has the following features:

- Contains eight cache-line (8x32 B) buffers to allow streaming of PCI transactions
- Performs address translation on outbound PCI transactions

28.7 PCI_IOS Memory Map and Register Definition

Table 28-3 shows the PCI memory map.

Table 28-42. Block Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x00	PCI Outbound Translation Address Register 0 (POTAR0)	R/W	28.7.1.1/28-63
0x08	PCI Outbound Base Address Register 0 (POBAR0)	R/W	28.7.1.2/28-64
0x10	PCI Outbound Comparison Mask Register 0 (POCMR0)	R/W	28.7.1.3/28-65
0x18	PCI Outbound Translation Address Register 1 (POTAR1)	R/W	28.7.1.1/28-63
0x20	PCI Outbound Base Address Register 1 (POBAR1)	R/W	28.7.1.2/28-64
0x28	PCI Outbound Comparison Mask Register 1 (POCMR1)	R/W	28.7.1.3/28-65
0x30	PCI Outbound Translation Address Register 2 (POTAR2)	R/W	28.7.1.1/28-63
0x38	PCI Outbound Base Address Register 2 (POBAR2)	R/W	28.7.1.2/28-64
0x40	PCI Outbound Comparison Mask Register 2 (POCMR2)	R/W	28.7.1.3/28-65
0x48	PCI Outbound Translation Address Register 3 (POTAR3)	R/W	28.7.1.1/28-63
0x50	PCI Outbound Base Address Register 3 (POBAR3)	R/W	28.7.1.2/28-64
0x58	PCI Outbound Comparison Mask Register 3 (POCMR3)	R/W	28.7.1.3/28-65
0x60	PCI Outbound Translation Address Register 4 (POTAR4)	R/W	28.7.1.1/28-63
0x68	PCI Outbound Base Address Register 4 (POBAR4)	R/W	28.7.1.2/28-64
0x70	PCI Outbound Comparison Mask Register 4 (POCMR4)	R/W	28.7.1.3/28-65
0x78	PCI Outbound Translation Address Register 5 (POTAR5)	R/W	28.7.1.1/28-63
0x80	PCI Outbound Base Address Register 5 (POBAR5)	R/W	28.7.1.2/28-64
0x88	PCI Outbound Comparison Mask Register 5 (POCMR5)	R/W	28.7.1.3/28-65
0xF0	Power Management Control Register (PMCR)	R/W	28.7.1.4/28-67
0xF8	Discard Timer Control Register (DTCR)	R/W	28.7.1.5/28-68

28.7.1 Register Descriptions

28.7.1.1 PCI Outbound Translation Address Registers (POTARn)_

The PCI outbound translation address register defines the location of the outbound translation window in the PCI (translated) address space.

Offset 0x00, 0x18, 0x30, 0x48, 0x60, 0x78

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													TA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-51. PCI Outbound Translation Address Registers (POTARn)

Table 28-43. POTARn Field Descriptions

Field	Description
TA	Translation Address. This field contains the starting address of the outbound translated address. This field corresponds to the most-significant 20 bits of a 32-bit address.

28.7.1.2 PCI Outbound Base Address Registers (POBARn)

The PCI outbound base address register defines the location of the outbound translation window in the local (source) memory space.

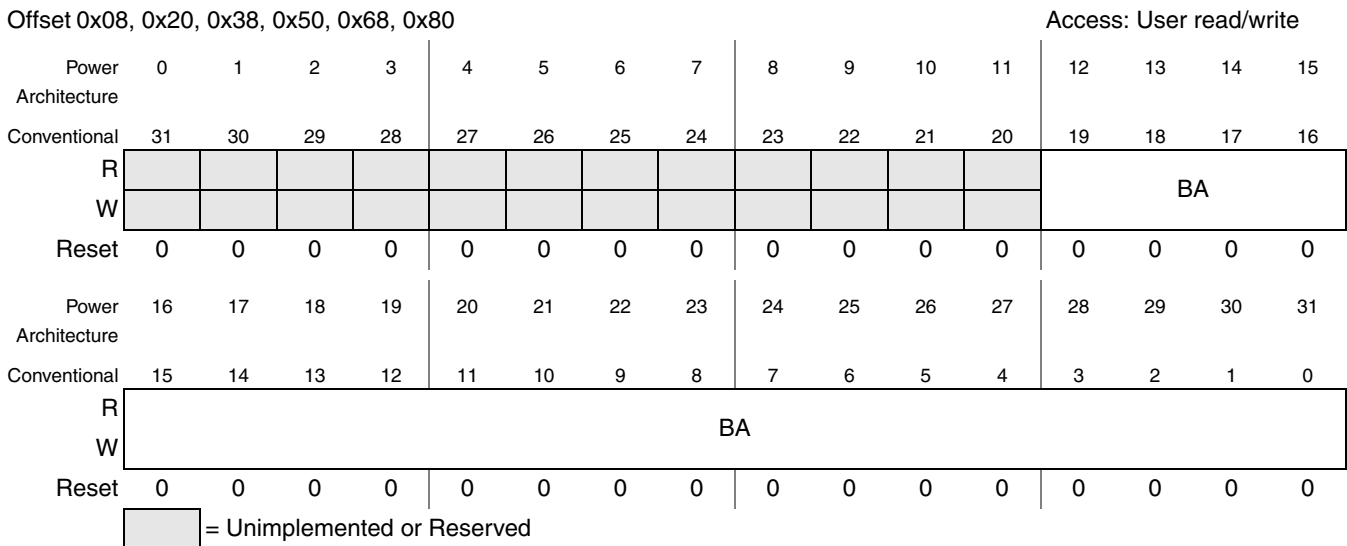


Figure 28-52. Outbound Base Address Registers (POBARn)

Table 28-44. POBARn Field Descriptions

Field	Description
BA	Base Address. This field contains the starting address in the local memory space before outbound translation. This field corresponds to the most-significant 20 bits of a 32-bit address.

28.7.1.3 PCI Outbound Comparison Mask Registers (POCMRn)

The PCI outbound comparison mask register defines the size and destination of the outbound translation window. It also defines some properties of the window in the PCI address space.

Offset 0x10, 0x28, 0x40, 0x58, 0x70, 0x88

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN	IO	PRES									SBS	CM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CM															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-53. PCI Outbound Comparison Mask Registers (POCMRn)

Table 28-45. POCMRn Field Descriptions

Field	Description
EN	Enable. This bit enables the address translation window. 0 Address translation is disabled for this window 1 Address translation is enabled for this window. Local addresses that match the definition of the window are recognized by the IOS and translated to the PCI memory space
IO	I/O Space. This bit determines whether the window is mapped to PCI memory space or PCI I/O space. 0 Memory space 1 I/O space
PRES	Prefetchable. This bit defines whether the transactions translated through this window are prefetchable on the PCI bus. Streaming the transactions requires the address space to be prefetchable. 0 Not prefetchable 1 Prefetchable
SBS	Special Byte Swap. This bit indicates that the data of transactions passing through this window should be subject to a special byte swap: each 32-bit half of the data word is 4-byte swapped. 0 Normal byte swap 1 Special byte swap
CM	Comparison Mask. This field determines the bits compared in the address, which effectively determines the size of the address translation window. See Figure 28-46 .

Table 28-46. CM Values

Value	Meaning
0000_0000_0000_0000_0000	4 GB
1000_0000_0000_0000_0000	2 GB
1100_0000_0000_0000_0000	1 GB
1110_0000_0000_0000_0000	512 MB
1111_0000_0000_0000_0000	256 MB
1111_1000_0000_0000_0000	128 MB
1111_1100_0000_0000_0000	64 MB
1111_1110_0000_0000_0000	32 MB
1111_1111_0000_0000_0000	16 MB
1111_1111_1000_0000_0000	8 MB
1111_1111_1100_0000_0000	4 MB
1111_1111_1110_0000_0000	2 MB
1111_1111_1111_0000_0000	1 MB
1111_1111_1111_1000_0000	512 Kbytes
1111_1111_1111_1100_0000	256 Kbytes
1111_1111_1111_1110_0000	128 Kbytes
1111_1111_1111_1111_0000	64 Kbytes
1111_1111_1111_1111_1000	32 Kbytes
1111_1111_1111_1111_1100	16 Kbytes
1111_1111_1111_1111_1110	8 Kbytes
1111_1111_1111_1111_1111	4 Kbytes
Others	Reserved

28.7.1.4 Power Management Control Register (PMCR)

This register provides control of system-level low-power modes.

Offset 0xF0

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																IDLE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1


 = Unimplemented or Reserved

Figure 28-54. Power Management Control Register (PMCR)

Table 28-47. PMCR Field Descriptions

Field	Description
IDLE	This read-only bit indicates logic controlled by the IOS power management function is idle. This bit could be used as an indication that it is OK to disable the clocks to this complex. 0 Logic is active 1 Logic is idle. There are no outstanding transactions in the IOS, the DMA, or the PCI port.

28.7.1.5 Discard Timer Control Register (DTCR)

This register configures the discard timer used to put a time limit on PCI delayed read transactions from non-prefetchable memory.

Offset 0xF8

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EN								PTV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PTV															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-55. Discard Timer Control Register (DTCR)

Table 28-48. DTCR Field Descriptions

Field	Description
EN	Enable. This bit enables the discard timer. 0 Disabled 1 Enabled
PTV	Preload Timer Value. This field contains the preload value for the discard timer. PCI delayed reads from non-prefetchable address space are discarded after $(2^{24} - \text{PTV})$ internal clock cycles if the master has not repeated the transaction. 0xFFFFF is not valid for PTV. Example: To discard a delayed completion if the PCI master has not repeated the transaction in 2^{15} PCI clocks, assuming the internal frequency is twice the PCI frequency, the preload timer value should equal $2^{24} - 2^{16}$ (0xFF0000).

28.8 Functional Description

The I/O Sequencer (IOS) is a 3-port switch with buffering. Each port has master and slave interfaces. When a port masters a transaction, the transaction attributes are stored in a buffer, and the IOS generates a transaction to the slave interface of the destination port. The data is also buffered between the ports. The IOS contains eight cache line (32-byte) transaction buffers, some of which are reserved for specific types of transactions.

The address and data phases of the transactions are independent. The data phases of the transactions are not required to be in-order.

28.8.1 Transaction Forwarding

Although the ports use a similar interface, the PCI_IOS is not actually symmetrical. The transaction forwarding from each source is explained in the following sections.

28.8.1.1 Transactions from the Local Port

Transactions from the local port are forwarded as follows:

- If the address matches the PCI controller software configuration memory space of the PCI controller, the transaction is forwarded to PCI port. These address values are configuration options of the PCI_IOS.
- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- If the address hits any of the outbound translation windows, the transaction is forwarded to PCI port, with the address translated.

28.8.1.2 Transactions from the PCI Port

Transactions from the PCI port are forwarded as follows:

- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- All other transactions are forwarded to the local port.

28.8.1.3 Transactions from the DMA Port

Transactions from the DMA port are forwarded as follows:

- If the address hits any of the outbound translation windows, the transaction is forwarded to PCI port, with the address translated.
- All other transactions are forwarded to the local port.

28.8.2 PCI Outbound Address Translation

Outbound address translation is provided to allow the outbound transactions to access any address over the PCI memory or I/O space. Translation window base addresses are defined in the PCI outbound base address registers. Transactions to these address ranges are issued on the PCI bus with a translated address. The translation addresses are defined in the associated PCI outbound translation address registers (POTARs). [Figure 28-56](#) shows an example translation window for outbound memory accesses.

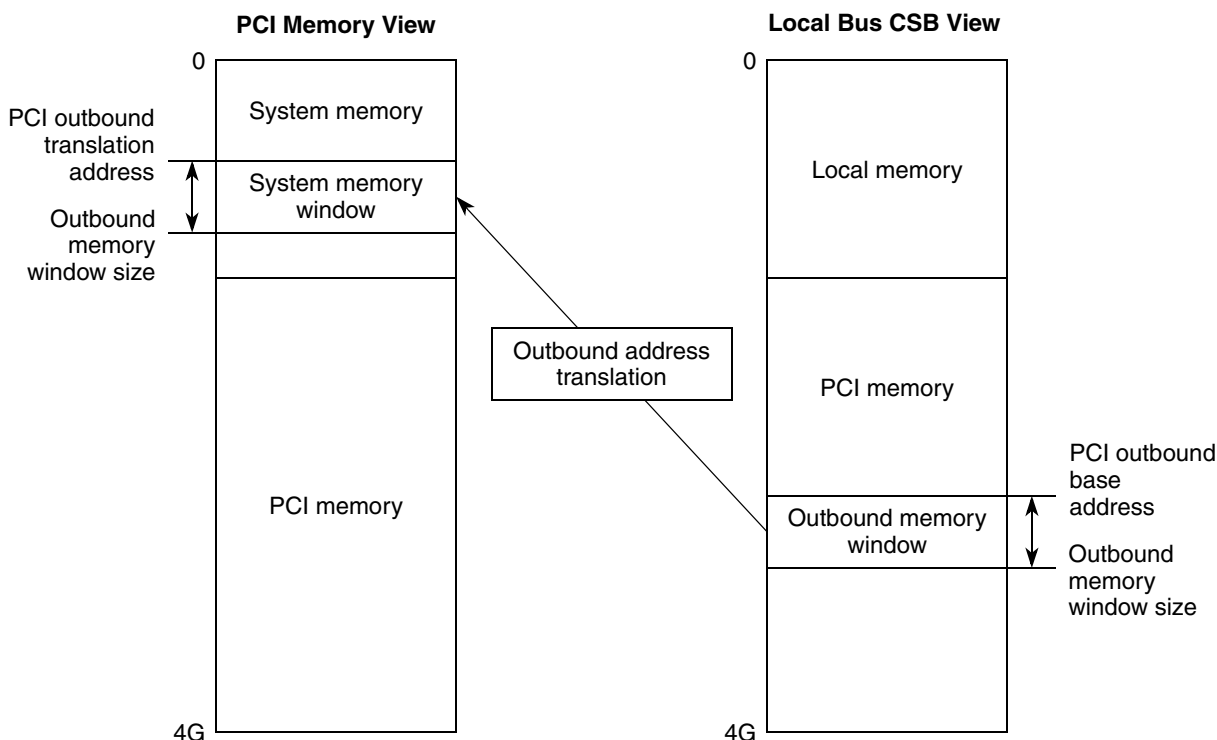


Figure 28-56. Outbound PCI Memory Address Translation

The six sets of outbound translation registers allow six simultaneous translation windows to the PCI port. Software can move and adjust the memory window translations and sizes during run-time. This allows software to access host memory or to address alternate memory space on the fly, but the PCI outbound translation windows may not overlap.

28.8.3 Transaction Ordering

The following rules are applied to maintain proper ordering of transactions:

- The transactions arriving from each port are dispatched to the destination port in the order of arrival. The dispatch order of transactions arriving on different ports is not necessarily maintained.
- Round-robin arbitration is used when transactions on different ports are occurring at the same time.

A read transaction that originates at the local port and reads from a PCI port pulls out of the IOS any posted writes that originated on the PCI port and were posted before the read data arrives from the PCI.

28.9 DMA for PCI Subsystem (PCI)

Figure 28-57 is a block diagram of the PCI.

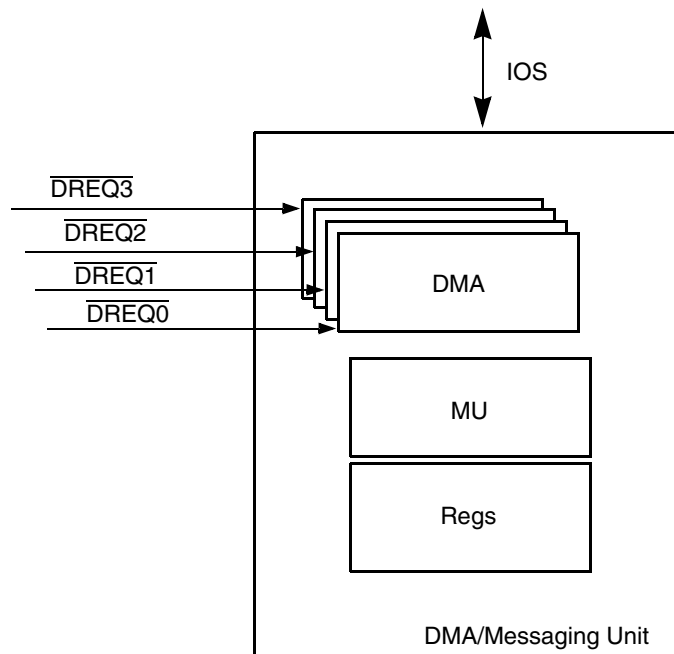


Figure 28-57. PCI_DMA Block Diagram

The PCI supports communication between two processors on different buses, e.g. a local processor and a processor on a PCI bus. This communication unit operates with generic messages and door bell registers.

The DMA controller that transfers blocks of data independent of the local core or PCI hosts. The DMA module has four high-speed DMA channels. The channels share 144 bytes of DMA-dedicated buffer space to facilitate the gathering and sending of data.

28.9.1 Features

The PCI includes the following features:

- Provides message and doorbell registers for inter-processor communication
- DMA Controller
 - Four DMA channels
 - Concurrent execution across multiple channels with programmable bandwidth control
 - Unaligned transfer capability
 - Data chaining and direct mode
 - Interrupt on completed segment, chain, and error

28.9.2 Modes of Operation

The DMA controller can operate in two modes—chaining and direct. The detail description is in [Section 28.12.2.1, “DMA Operation”](#).

28.10 External Signal Description

This section describes the DMA signals.

28.10.1 Detailed Signal Descriptions

[Table 28-49](#) contains the detailed descriptions of the DMA interface signals.

Table 28-49. DMA Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description
$\overline{\text{DREQ}}_{0-3}$	I	DMA Request. The DMA request signal indicates the start or continuation of a DMA transfer. The falling edge of $\overline{\text{DREQ}}_n$ causes $\text{DMAMRn}[\text{CS}]$ to be set, thereby activating the DMA channel.
		State Meaning Asserted—Assertion of $\overline{\text{DREQ}}_n$ starts or resumes a DMA transfer if $\text{DMAMRn}[\text{EMSEN}]$ is 1. Negated—Negation of $\overline{\text{DREQ}}_n$ has no effect.
		Timing Assertion—Can be asserted asynchronously Negation—Must remain asserted for at least one clock period.

28.11 Memory Map and Register Definition

[Table 28-50](#) shows the PCI memory map.

Table 28-50. Block Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x030	Outbound Message Interrupt Status Register (OMISR)	Special	28.11.1.1.1/28-74
0x034	Outbound Message Interrupt Mask Register (OMIMR)	R/W	28.11.1.1.2/28-75
0x050	Inbound Message Register 0 (IMR0)	R/W	28.11.1.2.1/28-76
0x054	Inbound Message Register 1 (IMR1)	R/W	28.11.1.2.1/28-76
0x058	Outbound Message Register 0 (OMR0)	R/W	28.11.1.2.2/28-76
0x05C	Outbound Message Register 1 (OMR1)	R/W	28.11.1.2.2/28-76
0x060	Outbound Doorbell Register (ODR)	R/W	28.11.1.3.1/28-77
0x068	Inbound Doorbell Register (IDR)	R/W	28.11.1.3.2/28-78
0x080	Inbound Message Interrupt Status Register (IMISR)	R/W	28.11.1.4.1/28-79
0x084	Inbound Message Interrupt Mask Register (IMIMR)	R/W	28.11.1.4.2/28-80
0x100	DMA 0 Mode Register (DMAMR0)	R/W	28.11.1.5.1/28-81
0x104	DMA 0 Status Register (DMASR0)	R/W	28.11.1.6/28-84

Table 28-50. Block Memory Map (continued)

Offset or Address	Register	Access	Section/Page
0x108	DMA 0 Current Descriptor Address Register (DMACDAR0)	R/W	28.11.1.6.1/28-85
0x110	DMA 0 Source Address Register (DMASAR0)	R/W	28.11.1.6.2/28-86
0x118	DMA 0 Destination Address Register (DMADAR0)	R/W	28.11.1.6.3/28-86
0x120	DMA 0 Byte Count Register (DMABCR0)	R/W	28.11.1.6.4/28-87
0x124	DMA 0 Next Descriptor Address Register (DMANDAR0)	R/W	28.11.1.6.5/28-88
0x180	DMA 1 Mode Register (DMAMR1)	R/W	28.11.1.5.1/28-81
0x184	DMA 1 Status Register (DMASR1)	R/W	28.11.1.6/28-84
0x188	DMA 1 Current Descriptor Address Register (DMACDAR1)	R/W	28.11.1.6.1/28-85
0x190	DMA 1 Source Address Register (DMASAR1)	R/W	28.11.1.6.2/28-86
0x198	DMA 1 Destination Address Register (DMADAR1)	R/W	28.11.1.6.3/28-86
0x1A0	DMA 1 Byte Count Register (DMABCR1)	R/W	28.11.1.6.4/28-87
0x1A4	DMA 1 Next Descriptor Address Register (DMANDAR1)	R/W	28.11.1.6.5/28-88
0x200	DMA 2 Mode Register (DMAMR2)	R/W	28.11.1.5.1/28-81
0x204	DMA 2 Status Register (DMASR2)	R/W	28.11.1.6/28-84
0x208	DMA 2 Current Descriptor Address Register (DMACDAR2)	R/W	28.11.1.6.1/28-85
0x210	DMA 2 Source Address Register (DMASAR2)	R/W	28.11.1.6.2/28-86
0x218	DMA 2 Destination Address Register (DAR2)	R/W	28.11.1.6.3/28-86
0x220	DMA 2 Byte Count Register (DMABCR2)	R/W	28.11.1.6.4/28-87
0x224	DMA 2 Next Descriptor Address Register (DMANDAR2)	R/W	28.11.1.6.5/28-88
0x280	DMA 3 Mode Register (DMAMR3)	R/W	28.11.1.5.1/28-81
0x284	DMA 3 Status Register (DMASR3)	R/W	28.11.1.6/28-84
0x288	DMA 3 Current Descriptor Address Register (DMACDAR3)	R/W	28.11.1.6.1/28-85
0x290	DMA 3 Source Address Register (DMASAR3)	R/W	28.11.1.6.2/28-86
0x298	DMA 3 Destination Address Register (DMADAR3)	R/W	28.11.1.6.3/28-86
0x2A0	DMA 3 Byte Count Register (DMABCR3)	R/W	28.11.1.6.4/28-87
0x2A4	DMA 3 Next Descriptor Address Register (DMANDAR3)	R/W	28.11.1.6.5/28-88
0x2A8	DMA General Status Register (DMAGSR)	R	28.11.1.6.6/28-89

28.11.1 Register Descriptions

28.11.1.1 Outbound Message Interrupt Registers

28.11.1.1.1 Outbound Message Interrupt Status Register (OMISR)

OMISR contains the interrupt status of the door bell and outbound message registers. A PCI device acknowledges the outbound message interrupt by writing a 1 to the appropriate status bit, OMISR[OM1I] or OMISR[OMI0]. This clears the interrupt and the corresponding status bit. The local processor provokes an outbound message interrupt by writing to either of the two outbound message registers, OMR0 or OMR1. MISR should be accessed only from the PCI bus.

Offset 0x30												Access: Special				
Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													ODI			
W															OM1I	OM0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 28-58. Outbound Message Interrupt Status Register (OMISR)

Table 28-51. OMISR Field Descriptions

Field	Description
ODI	Outbound Doorbell Interrupt. When set, indicates that there is an outbound doorbell interrupt. 0 No outbound doorbell interrupt 1 There is an outbound doorbell interrupt
OM1I	Outbound Message 1 Interrupt. When set, indicates there is an outbound message 1 interrupt. Write 1 to this position to clear this bit. 0 No Outbound Message 1 interrupt 1 There is an Outbound Message 1 interrupt
OMI0	Outbound Message 0 Interrupt. When set, indicates there is an outbound message 0 interrupt. Write 1 to this position to clear this bit. 0 No Outbound Message 0 interrupt 1 There is an Outbound Message 0 interrupt

28.11.1.1.2 Outbound Message Interrupt Mask Register (OMIMR)

OMIMR contains the interrupt mask of the door bell and message register events generated by the local processor. OMIMR should be accessed only from the PCI bus.

Offset 0x34

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													ODIM		OM1IM	OM0IM
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 28-59. Outbound Message Interrupt Mask Register (OMIMR)

Table 28-52. OMIMR Field Descriptions

Field	Description
ODIM	Outbound Doorbell Interrupt Mask. 0 Outbound doorbell interrupt is allowed 1 Outbound doorbell interrupt is masked
OM1IM	Outbound Message 1 Interrupt Mask. 0 Outbound Message 1 interrupt is allowed 1 Outbound Message 1 interrupt is masked
OM0IM	Outbound Message 0 Interrupt Mask. 0 Outbound Message 0 interrupt is allowed 1 Outbound Message 0 interrupt is masked

28.11.1.2 Message Registers

28.11.1.2.1 Inbound Message Registers (IMR0, IMR1)

The inbound message registers are accessible from the PCI bus and the local bus in host mode.

Offset 0x50, 0x54

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IMSGx															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IMSGx															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 28-53. Inbound Message Registers (IMR0, IMR1)

Table 28-54. IMR0 and IMR1 Field Descriptions

Field	Description
IMSGx	Inbound Message x. Contains generic data to be passed between the local processor and external hosts.

28.11.1.2.2 Outbound Message Registers (OMR0, OMR1)

The outbound message registers are accessible from the PCI bus and the local bus in host mode.

Offset 0x58, 0x5C

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OMSGx															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OMSGx															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-60. Outbound Message Registers (OMR0, OMR1)

Table 28-55. OMR0 and OMR1 Field Descriptions

Field	Description
OMSGx	Outbound Message x. Contains generic data to be passed between the local processor and external hosts.

28.11.1.3 Doorbell Registers

28.11.1.3.1 Outbound Doorbell Register (ODR)

ODR is accessible from the PCI bus and the local bus in host mode.

Offset 0x60

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR
W				28	27	26	25	24	23	22	21	20	19	18	17	16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR	ODR
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-61. Outbound Doorbell Register (ODR)

Table 28-56. ODR Field Descriptions

Field	Description
ODRx	Outbound Doorbell x. Write 1 from the local bus to set. Write 1 from the PCI bus to clear. Writing a bit in this register from the local bus causes an interrupt (\overline{INTA}) to be generated.

28.11.1.3.2 Inbound Doorbell Register (IDR)

IDR is accessible from the PCI bus and the local bus in host and mode.

Offset 0x68

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IMC	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR
W		30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR	IDR
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-62. Inbound Doorbell Register (IDR)

Table 28-57. IDR Field Descriptions

Field	Description
IMC	Inbound Machine Check. Write 1 from the PCI bus to set. Write 1 from the local bus to clear. Writing a bit in this register from the PCI bus causes a machine check interrupt to be generated to the local processor.
IDRx	Inbound Doorbell x. Write 1 from the PCI bus to set. Write 1 from the local bus to clear. Writing a bit in this register from the PCI bus causes an interrupt to be generated to the local processor.

28.11.1.4 Inbound Message Interrupt Registers

28.11.1.4.1 Inbound Message Interrupt Status Register (IMISR)

This register contains the interrupt status of the door bell and message register events. Writing a 1 to the corresponding set bit clears the bit. The events are generated by the PCI masters. IMISR should be accessed only from the local bus. Accesses while in host mode or from the PCI bus have undefined results.

Offset 0x80												Access: User read/write				
Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R												MCI	IDI		IM1I	IM0I
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 28-63. Inbound Message Interrupt Status Register (IMISR)

Table 28-58. IMISR Field Descriptions

Field	Description
MCI	Machine Check Interrupt. 0 No Machine Check interrupt 1 There is a Machine Check interrupt
IDI	Inbound Doorbell Interrupt. 0 No Inbound Doorbell interrupt 1 There is an Inbound Doorbell interrupt
IM1I	Inbound Message 1 Interrupt. 0 No Inbound Message 1 interrupt 1 There is an Inbound Message 1 interrupt
IM0I	Inbound Message 0 Interrupt. 0 No Inbound Message 0 interrupt 1 There is an Inbound Message 0 interrupt

28.11.1.4.2 Inbound Message Interrupt Mask Register (IMIMR)

This register contains the interrupt mask of the door bell and message register events generated by the PCI master. IMIMR should be accessed only from the local bus and only in agent mode. Accesses while in host mode or from the PCI bus have undefined results.

Offset 0x84

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													MCIM	IDIM		
W															IM1IM	IM0IM
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-64. Inbound Message Interrupt Mask Register (IMIMR)

Table 28-59. IMIMR Field Descriptions

Field	Description
MCIM	Machine Check Interrupt Mask. 0 Machine Check interrupt from the inbound doorbell register is allowed 1 Machine Check interrupt is masked
IDIM	Inbound Doorbell Interrupt Mask. 0 Inbound Doorbell interrupt is allowed 1 Inbound Doorbell interrupt is masked
IM1IM	Inbound Message 1 Interrupt Mask. 0 Inbound Message 1 interrupt is allowed 1 Inbound Message 1 interrupt is masked
IM0IM	Inbound Message 0 Interrupt Mask. 0 Inbound Message 0 interrupt is allowed 1 Inbound Message 0 interrupt is masked

28.11.1.5 DMA Registers

Each DMA channel has a set of seven 32-bit registers (mode, status, current descriptor address, next descriptor address, source address, destination address, and byte count) to support transactions. This section describes the format of the DMA support registers.

28.11.1.5.1 DMA Mode Register (DMAMRx)

The mode register allows software to start the DMA transfer and to control various DMA transfer characteristics.

Offset 0x100, 0x180, 0x200, 0x280

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					DRCNT				BWC			DMS EN	IRQS	EMS EN	DAHTS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SAHTS		DAHE	SAHE	PRC				EOTIE				TEM	CTM	CC	CS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 28-65. DMA Mode Register (DMAMRx)
(Register is repeated for reference.)

Table 28-60. DMAMRx Field Descriptions (Sheet 1 of 3)

Field	Description
DRCNT	DMA Request CouNt. This field specifies the number of cache lines transferred per DMA request assertion if EMSEN is 1. This field is not used if EMSEN is 0. 0101 1 cache line 0110 2 cache lines 0111 4 cache lines 1000 8 cache lines 1001 6 cache lines 1010 32 cache lines OthersReserved
BWC	Bandwidth Control. This field only applies when multiple channels are executing transfers concurrently. The field determines how many cache lines a given channel is allowed to transfer after it is granted access to the IOS interface and before it releases the interface to the next channel. This allows prioritization of the DMA channels. 000 1 cache line 001 2 cache lines 010 4 cache lines 011 8 cache lines 100 16 cache lines OthersReserved

PCI Controller (PCI)

Offset 0x100, 0x180, 0x200, 0x280

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					DRCNT				BWC			DMS EN	IRQS	EMS EN	DAHTS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SAHTS		DAHE	SAHE	PRC				EOTIE				TEM	CTM	CC	CS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 28-65. DMA Mode Register (DMAMRx)
(Register is repeated for reference.)

Table 28-60. DMAMRx Field Descriptions (Sheet 2 of 3)

Field	Description
DMSEN	Direct Mode Snoop Enable. This bit controls snooping of direct mode DMA transactions. 0 Snooping is disabled 1 Snooping is enabled
IRQS	Interrupt Steer. This bit determines the destination of the DMA interrupts. 0 All DMA interrupts are routed to the local processor 1 All DMA interrupts are routed to the PCI bus through \overline{INTA}
EMSEN	External master start enable. This bit is cleared when the DMA transfer has completed, so it must be set again for each transfer. 0 The channel is started by software setting the CS bit. 1 The channel is started by hardware asserting the DREQ pin.
DAHTS	Destination Address Hold Transfer Size. This field indicates the transfer size used for each transaction when DAHE is 1. The byte count register must be in multiples of the size, and the destination address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
SAHTS	Source Address Hold Transfer Size. This field indicates the transfer size used for each transaction when SAHE is 1. The byte count register must be in multiples of the size, and the source address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
DAHE	Destination Address Hold Enable. This bit allows the DMA controller to hold the destination address constant for every transfer. The size used for transfer is indicated by DAHTS. Hardware supports only aligned transfers for this feature. 0 Don't hold the destination address constant 1 Hold the destination address constant

Offset 0x100, 0x180, 0x200, 0x280

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					DRCNT				BWC			DMS EN	IRQS	EMS EN	DAHTS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SAHTS		DAHE	SAHE	PRC				EOTIE				TEM	CTM	CC	CS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 28-65. DMA Mode Register (DMAMRx)
(Register is repeated for reference.)

Table 28-60. DMAMRx Field Descriptions (Sheet 3 of 3)

Field	Description
SAHE	Source Address Hold Enable. This bit allows the DMA controller to hold the source address constant for every transfer. The size used for transfer is indicated by SAHTS. Hardware supports only aligned transfers for this feature. 0 Don't hold the source address constant 1 Hold the source address constant
PRC	PCI Read Command. This field indicates the type of PCI read command to use. 00 PCI read 01 PCI Read Line 10 PCI Read Multiple 11 Reserved
EOTIE	End-of-Transfer Interrupt Enable. This bit determines whether an interrupt is generated at the completion of a DMA transfer. End-of-transfer is defined as the end of a direct mode transfer or in chaining mode as the end of the transfer of the last segment of a chain. 0 No EOT interrupt is generated 1 EOT interrupt is generated
TEM	Transfer Error Mask. This bit determines the DMA response in the event of a transfer error. 0 The DMA halts when a transfer error occurs (TE bit is set) 1 The DMA completes the transfer regardless of whether a transfer error occurs (the TE bit is not set)
CTM	Channel Transfer Mode 0 Chaining Mode 1 Direct Mode
CC	Channel Continue. This bit applies only to chaining mode and is cleared by hardware after every descriptor read. 0 The DMA transfer does not restart the transferring process 1 The DMA transfer restarts the transferring process starting at the current descriptor address
CS	Channel Start. A 0-to-1 transition occurring on this bit when the channel is not busy (SR[CB] bit is 0) starts the DMA process. If the channel is busy and a 0 to 1 transition occurs, the DMA channel restarts from a previous halt condition. A 1-to-0 transition when the channel is busy (CB bit is 1) halts the DMA process. Nothing happens if the channel is not busy and a 1 to 0 transition occurs.

28.11.1.6 DMA Status Register (DMASRn)

The status register reports various DMA conditions during and after the DMA transfer. Writing a 1 to a specific set bit clears the bit.

Offset 0x104, 0x184, 0x204, 0x284

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									TE					CB	EOSI	EOCDI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 28-66. DMA Status Register (DMASRn)

Table 28-61. DMASRn Field Descriptions

Field	Description
TE	Transfer Error. This bit is set when there is an error condition during the DMA transfer.
CB	Channel Busy. This bit indicates whether the channel is busy. It is cleared as a result of an error or completion of the DMA transfer. 0 No DMA transfer is currently in progress 1 A DMA transfer is currently in progress
EOSI	End-of-Segment Interrupt. After transferring a segment of data, if the EOSIE bit in the current descriptor address register is set, this bit is set and an interrupt is generated.
EOCDI	End-of-Chain/Direct Interrupt. When the last DMA transfer is finished, either in chaining or direct mode, if DMAMR[EOTIE] is set, this bit is set and an interrupt is generated.

28.11.1.6.1 DMA Current Descriptor Address Register (DMACDARn)

The current descriptor address register contains the address of the current segment descriptor being transferred. In chaining mode, software must initialize this register to point to the first descriptor in the chain. After processing the first descriptor, the DMA controller moves the contents of the next descriptor address register into DMACDAR, loads the following descriptor into DMANDAR, and executes the current transfer.

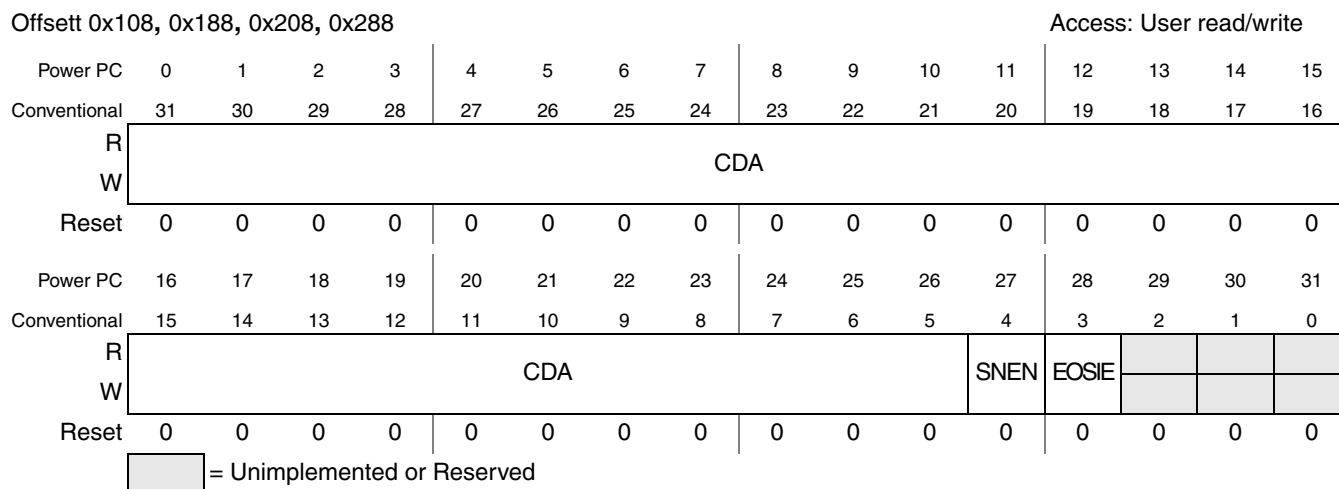


Figure 28-67. DMA Current Descriptor Address Register (DMACDARn)

Table 28-62. DMACDARn Field Descriptions

Field	Description
CDA	Current Descriptor Address. This field contains the current descriptor address of the segment descriptor in memory. It must be aligned on an 8-word boundary
SNEN	Snoop Enable. 0 Snooping is disabled on DMA transactions 1 Snooping is enabled on DMA transactions
EOSIE	End-of-Segment Interrupt Enable. 0 No end-of-segment interrupt is generated 1 An interrupt is generated when the current DMA transfer for the current descriptor is finished

28.11.1.6.2 DMA Source Address Register (DMASARn)

The source address register indicates the address the DMA controller reads data from. The software must ensure this is a valid memory address.

Offset 0x110, 0x190, 0x210, 0x290

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-68. DMA Source Address Register (DMASARn)

Table 28-63. DMASARn Field Descriptions

Field	Description
SA	Source Address of DMA Transfer. The content of this field is updated after each DMA read operation.

28.11.1.6.3 DMA Destination Address Register (DMADARn)

The destination address register indicates the address the DMA controller writes data to. The software must ensure this is a valid memory address.

Offset 0x118, 0x198, 0x218, 0x298

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-69. DMA Destination Address Register (DMADARn)

Table 28-64. DMADARn Field Descriptions

Field	Description
DA	Destination Address of DMA Transfer. The content of this field is updated after each DMA write operation.

28.11.1.6.4 DMA Byte Count Register (DMABCRn)

This register contains the number of bytes per transfer (maximum transfer size is 64 Mbytes).

Offset 0x120, 0x1A0, 0x220, 0x2A0

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							BC									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BC															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-70. DMA Byte Count Register (DMABCRn)

Table 28-65. DMABCRn Field Descriptions

Field	Description
BC	Byte Count. This field contains the number of bytes to transfer. The value in this register is decremented after each DMA read operation.

28.11.1.6.5 DMA Next Descriptor Address Register (DMANDARn)

The next descriptor address register (NDAR) contains the address for the next segment descriptor in the chain. In chaining mode, this register is loaded from the next descriptor field of the descriptor the current descriptor address register points to.

Offset 0x124, 0x1A4, 0x224, 0x2A4

Access: User read/write

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	NDAR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NDAR												NSNEN	NEOSIE		EOTD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-71. DMA Next Descriptor Address Register (DMANDARnx)

Table 28-66. DMANDARn Field Descriptions

Field	Description
NDA	Next Descriptor Address. This field contains the next descriptor address of the segment descriptor in memory. It must be aligned on an 8-word boundary.
NSNEN	Next Snoop Enable 0 Snooping is disabled on DMA transactions 1 Snooping is enabled on DMA transactions
NEOSIE	Next End-of-Segment Interrupt Enable 0 No end-of-segment interrupt is generated 1 An interrupt is generated when the DMA transfer for the next descriptor is finished
EOTD	End-of-Transfer Descriptor 0 This descriptor contains a link to another descriptor 1 This descriptor is the last to be executed

28.11.1.6.6 DMA General Status Register (DMAGSR)

This read-only register provides faster access to the status bits by combining the status bits of all of the DMA channels into one register. Each byte of this register provides the value of bits 24:31 of a channel's DMA status register. These bits are cleared by writing to the individual DMA status registers.

Offset 0x2A8 Access: User read only

Power PC	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Channel 0 Status								Channel 1 Status							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power PC	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Channel 2 Status								Channel 3 Status							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 28-72. DMA General Status Register (DMAGSR)

28.12 Functional Description

28.12.1 Message Unit

An embedded processor is often part of a larger system containing many processors and distributed memory. These processors tend to work on tasks independent of the host processor(s) and other peripheral processors in the system. Because of the independent nature of the tasks, it is necessary to provide a communication mechanism between the peripheral processors and the rest of the system. One such method is the use of messages. This block provides a messaging unit to further facilitate communications between host and peripheral. The message unit uses generic messages and doorbell registers.

28.12.1.1 Message Passing

This block contains two inbound message registers and two outbound message registers. The registers are each 32 bits. The inbound registers, Inbound Message Register 0 (IMR0) and Inbound Message Register 1 (IMR1), allow a remote host or PCI master to write a 32-bit value that causes an interrupt to the local processor. This implements the PowerPC architecture because the register indirectly drives an interrupt line to the local processor. The outbound registers, Outbound Message Register 0 (OMR0) and Outbound Message Register 1 (OMR1), allow the local processor to write an outbound message that causes the outbound interrupt signal \overline{INTA} to assert.

The interrupt to the local processor is cleared by setting the appropriate bit in the Inbound Message Interrupt Status Register (IMISR). The interrupt to PCI (\overline{INTA}) is cleared by setting the appropriate bit in the Outbound Message Interrupt Status Register (OMISR).

28.12.1.2 Doorbells

This block contains the Inbound Doorbell Register (IDR) and the Outbound Doorbell Register (ODR). The inbound doorbell allows a remote processor to set a bit in the register from the PCI bus. This causes the PCI bridge to generate an interrupt to the local processor. The local processor can write to the outbound register. This causes the outbound interrupt signal, \overline{INTA} , to assert and interrupt the remote processor on the PCI bus.

28.12.2 DMA Controller

The DMA controller transfers blocks of data independent of the local core or PCI hosts. Data movement occurs on the PCI and/or local bus. The DMA module has four high-speed DMA channels. The channels share 144 bytes of DMA-dedicated buffer space to facilitate the gathering and sending of data. The local core and PCI masters can initiate a DMA transfer.

Features of the DMA controller include:

- Four channels
- Concurrent execution across multiple channels with programmable bandwidth control
- All channels are accessible by local core and remote PCI masters.
- Unaligned transfer capability
- Data chaining and direct mode
- Interrupt on completed segment, chain, and error

Figure 28-73 shows a diagram of the DMA controller in a representative system.

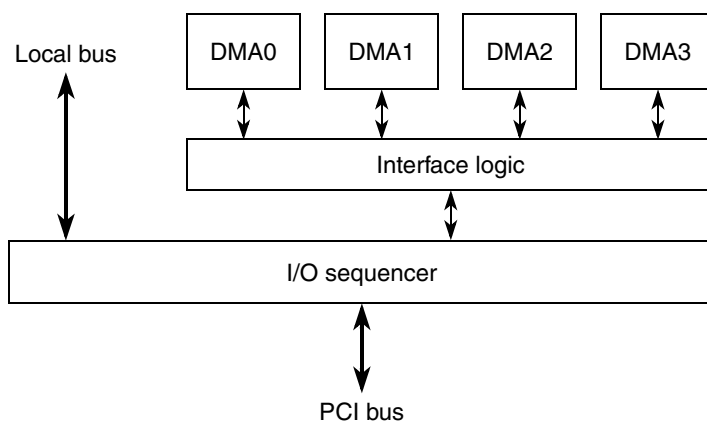


Figure 28-73. DMA Controller Block Diagram

28.12.2.1 DMA Operation

The DMA controller operates in two modes—chaining and direct. In direct mode, the software is responsible for initializing the source, destination and byte count registers. In chaining mode, the software must first build a chain of descriptor segments in external memory, residing on the local or PCI bus, and then initialize the descriptor, DMA Current Descriptor Address Register (DMACDARn), to point to the

first descriptor segment in the chain. In both modes, setting the start bit in the DMA mode register begins the DMA transfer.

The DMA controller supports unaligned transfers for the source and destination addresses. It gathers data beginning at the source address and aligns the data accordingly before sending it to the destination address. The DMA controller assumes the source and destination addresses are valid PCI or local memory addresses.

All local memory read operations are cache line reads (32 bytes); the DMA controller selects the appropriate/valid data bytes within the cache line when loading its internal queue. Writing to local memory depends on the alignment of the destination address and the size of the transfer. The DMA controller writes a full cache line when possible. Misaligned destination addresses result in sub-transfers of less than a cache line on the initial and final beats of the transfer; intermediate beats transfer full cache lines. Configuring a DMA channel for a transfer size of less than 8 bytes in address hold mode, DAHE or SAHE is set in the DMA Mode Register (DMAMRx), precludes cache line writes.

PCI memory read operations depend on the PRC (PCI read command) field in the mode register, the alignment of the source address and the size of the transfer. The DMA controller attempts to read a full cache line whenever possible. Writing to PCI memory depends on the alignment of the destination address and the size of the transfer.

28.12.2.1.1 DMA Direct Mode

In direct mode, the DMA controller does not read a chain of descriptors from memory, but uses the current parameters in the DMA registers to start a DMA transfer. The DMA transfer finishes after all the bytes specified in the byte count register have been transferred or an error condition has occurred.

Below are the initialization steps of a DMA transfer in direct mode.

- Poll the CB (channel busy) bit in the DMA Status Register (DMASRn) to make sure the DMA channel is idle.
- Initialize the DMA Source Address Register (DMASARn), the DMA Destination Address Register (DMADARn), and the DMA Byte Count Register (DMABCRn).
- Initialize the CTM (channel transfer mode) bit in the DMA Mode Register (DMAMRx) to indicate direct mode. Other control parameters in the mode register can also be initialized here if necessary.
- Clear and then set the CS (channel start) bit in the DMA Mode Register (DMAMRx) to start the DMA transfer.

28.12.2.1.2 DMA Chaining Mode

In chaining mode, the DMA controller loads descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the descriptor information loaded for each segment. After the current segment is finished, the DMA controller reads the next descriptor from memory and begins another DMA transfer. The process is finished if the current descriptor is the last one in the chain or an error condition has occurred.

Below are the initialization steps of a DMA transfer in chaining mode.

- Build a chain of descriptor segments in memory. Refer to [Section 28.12.2.2, “DMA Segment Descriptors”](#) for more information.
- Poll the CB (channel busy) bit in the DMA Status Register (DMASRn) to make sure the DMA channel is idle.
- Initialize the DMA Current Descriptor Address Register (DMACDARn) to point to the first descriptor in the chain.
- Initialize the CTM (channel transfer mode) bit in the DMA Mode Register (DMAMRx) to indicate chaining mode. Other control parameters in the mode register can also be initialized here if necessary.
- Clear and then set the CS (channel start) bit in the DMA Mode Register (DMAMRx) to start the DMA transfer.

28.12.2.1.3 External Control

The DMA transfer of any channel, in direct or chaining mode, can be controlled by the DMA request input signals. External control is enabled by setting DMAMRn[EMSEN].

When using external control, the software should not set DMAMRn[CS]. A falling edge on DREQn sets DMAMRn[CS] to start the transfer. The corresponding DACKn output signals are asserted. The number of cache lines specified by DMAMRn[DRCNT] is transferred, and then DMAMRn[CS] is negated to halt the transfer until the next DREQn assertion. The corresponding DACKn output signals is negated. DREQn may be negated after DACKn is asserted or when the first transaction of the DMA transfer appears on the external interface, and it may be asserted again one clock cycle later. DDONEn is asserted when the DMA transfer has completed (all bytes specified in the byte count register or the descriptors have been transferred).

28.12.2.1.4 DMA Coherency

The four DMA channels uses up to four cache lines (128 bytes) of buffer space in the I/O sequencer module in addition to 16 bytes of local buffer space. Because no address snooping occurs in these internal queues, data posted in these queues is not visible to the rest of the system while a DMA transfer is in progress. It is the responsibility of application software to ensure the coherency of the region transferred during the DMA process.

Snooping of the core data cache is selectable during DMA transactions. A snoop bit is provided in the DMA Current Descriptor Address Register (DMACDARn) and the DMA Next Descriptor Address Register (DMANDARn), which allows software to control when the cache is snooped.

28.12.2.1.5 Halt and Error Conditions

DMA transfers are halted by clearing the CS (channel start) bit in the DMA Mode Register (DMAMRx) or when encountering an error condition. In both cases, the application software can do one of the following:

- Continue the DMA transfer
- Reconfigure the DMA for a new transfer
- Leave the channel in the halted state

When a DMA channel is halted, its programming model is completely accessible. If the DMA is halted due to an error condition, the TE (transfer error) bit in the DMA Status Register (DMASRn) must be cleared before the transfer can be resumed or a new transfer initiated. The TE bit is not cleared automatically by hardware.

NOTE

After any bus error occurs in the system (either local bus or PCIBus, not necessarily due to DMA operation), reset the system to avoid DMA malfunction.

28.12.2.2 DMA Segment Descriptors

DMA segment descriptors contain the source and destination addresses of the data segment, the segment byte count, and a link to the next descriptor. Segment descriptors are built on cache-line (32-byte) boundaries in local or PCI memory and are linked together into chains using the next-descriptor-address field.

Table 28-67. DMA Segment Descriptor Fields

Descriptor Field	Description
Source address	Contains the source address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the DMA Source Address Register (DMASARn).
Destination address	Contains the destination address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the DMA Destination Address Register (DMADARn).
Next descriptor address	Points to the next descriptor in memory. After the DMA controller reads the descriptor from memory, this field is loaded into the DMA Next Descriptor Address Register (DMANDARn).
Byte count	Contains the number of bytes to transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the DMA Byte Count Register (DMABCRn).

Application software initializes the current descriptor address register, DMA Current Descriptor Address Register (DMACDARn), to point to the first descriptor in the chain. For each descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by the descriptor. The DMA controller traverses the descriptor chain until reaching the last descriptor (with its EOTD bit set).

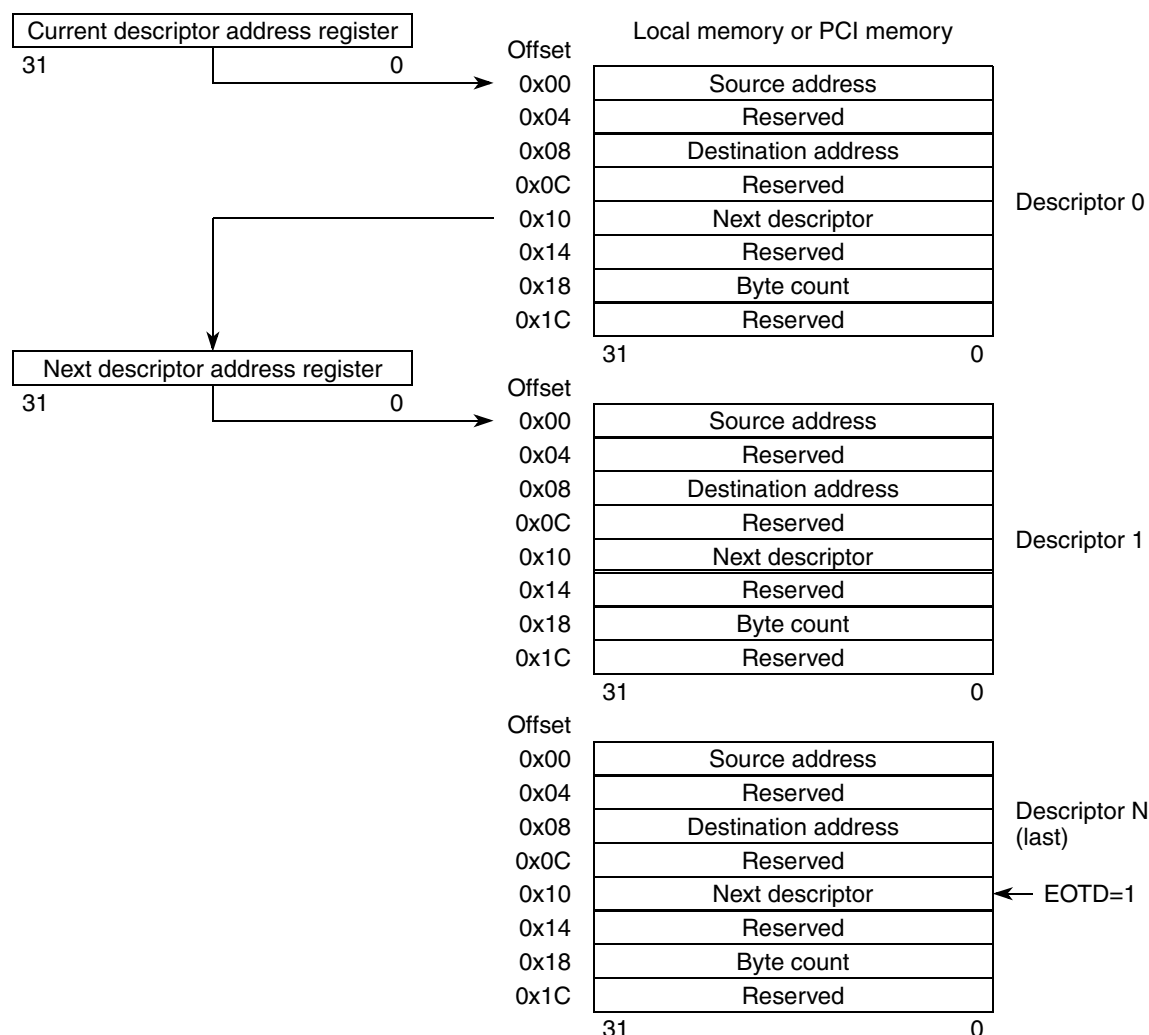


Figure 28-74. DMA Chain of Segment Descriptors

28.12.2.2.1 Descriptor in Big Endian Mode

In big endian mode, the descriptor in local memory should be programmed so data appears in ascending significant-byte order. If segment descriptors are written to memory located in the local bus, they should be treated like they are translated from big endian to little endian mode.

Example 28-1. Big Endian mode descriptor's data structure.
The descriptor structure must be aligned on an 8-word boundary.

```
struct {
    double a;          /* 0x1122334455667788 double word */
    double b;          /* 0x55667788aabbccdd double word */
    double c;          /* 0x8765432101234567 double word */
    double d;          /* 0x0123456789abcdef double word */
} Descriptor;
Results: Source Address = 0x44332211 <MSB..LSB>
        Destination Address = 0x88776655 <MSB..LSB>
        Next Descriptor Address = 0x21436587 <MSB..LSB>
        Byte Count = 0x67452301 <MSB..LSB>
```

28.12.2.2.2 Descriptor in Little Endian Mode

In little endian mode, the descriptor in PCI memory should be programmed so data appears in descending significant byte order. If segment descriptors are written to memory located in the PCI bus, they are obeying the rules for little endian mode.

Example 28-2. Little Endian mode descriptor's data structure.
The descriptor structure must be aligned on an 8-word boundary.

```

struct {
    double a;          /* 0x8877665544332211 double word */
    double b;          /* 0x1122334488776655 double word */
    double c;          /* 0x7654321012345678 double word */
    double d;          /* 0x0123456776543210 double word */
} Descriptor;
Results: Source Address = 0x44332211 <MSB..LSB>
        Destination Address = 0x88776655 <MSB..LSB>
        Next Descriptor Address = 0x12345678 <MSB..LSB>
Byte Count = 0x76543210 <MSB..LSB>

```


Chapter 29

Power Management Control Module (PMC)

29.1 Introduction

This document provides a description of the power management control (PMC) module. PMC is responsible for entering and exiting the low-power mode.

29.1.1 Features

The PMC module includes the following features:

- Causes the e300 CPU core to enter low-power mode (Nap or Sleep) after the coherent system bus (CSB) is idle and DRAM controller is idle
- Causes the DRAM controller to enter and exit low-power (self-refresh) mode
- Optional interrupt when exiting low-power mode (Nap or Sleep)
- Exit low-power mode when the CPU is ready
- Enter and exit Deep Sleep mode (system OSC, system PLL, and e300 core PLL are put in low-power standby mode)
- Support e300 core PLL on-the-fly change and system clock divide ratio (SYS_DIV) copy enable mode

29.2 Memory Map and Register Definition

Table 29-1. PMC Memory Map

Address Offset	Use	Access
0x0	PMC Configuration Register (PMCCR)	R/W
0x4	PMC Event Register (PMCER)	R/W
0x8	PMC Mask Register (PMCMR)	R/W
0xC	PMC CORE_PLL Shadow Register (PMCSR)	R/W

29.2.0.1 PMC Configuration Register (PMCCR)

Offset 0x0

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	PRE_DIV	CCM	DSM	DDROFF	CORE OFF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 29-1. PMC Configuration Register (PMCCR)

Table 29-2. PMCCR Field Descriptions

PRE DIV	CCM	DSM	DDR OFF	CORE OFF	Description
0	0	0	0	0	When PowerArchitecture core hits breakpoint, enter break mode When PowerArchitecture core initiates nap ¹ mode or sleep ² mode entry: undefined, reserved
0	0	0	0	1	When PowerArchitecture core hits breakpoint: undefined, reserved When PowerArchitecture core initiates nap mode entry: enter nap mode; do not put DRAM in self-refresh mode during nap mode. When PowerArchitecture core initiates sleep mode entry: enter sleep mode; do not put DRAM in self-refresh mode during sleep mode PMCCR is cleared after return to full power mode
0	0	0	1	1	When PowerArchitecture core hits breakpoint: undefined, reserved When PowerArchitecture core initiates nap mode entry: enter nap mode; ; put DRAM in self-refresh mode during nap mode When PowerArchitecture core initiates sleep mode entry: enter sleep mode; put DRAM in self-refresh mode during sleep mode PMCCR is cleared after return to full power mode
0	0	1	0	0	When PowerArchitecture core hits breakpoint: undefined, reserved When PowerArchitecture core initiates nap mode entry: undefined, reserved When PowerArchitecture core initiates sleep mode entry: enter deep sleep mode; PMCCR is cleared after return to full power mode

Table 29-2. PMCCR Field Descriptions

PRE DIV	CCM	DSM	DDR OFF	CORE OFF	Description
0	1	0	0	0	When PowerArchitecture core hits breakpoint: undefined, reserved When PowerArchitecture core initiates nap mode entry: undefined, reserved When PowerArchitecture core initiates sleep mode entry: enter core PLL change mode PMCCR is cleared after return to full power mode
1	0	0	0	0	When PowerArchitecture core hits breakpoint: undefined, reserved When PowerArchitecture core initiates nap mode entry: undefined, reserved When PowerArchitecture core initiates sleep mode entry: enter Pre divider copy mode PMCCR is cleared after return to full power mode
Any other combination					Undefined, reserved

¹ PowerArchitecture core initiates nap mode entry by setting the e300 NAP bit in the HID register while the POW bit in MSR register is set.

² PowerArchitecture core initiates sleep mode entry by setting the e300 SLEEP bit in the HID register while the POW bit in MSR register is set.

29.2.0.2 PMC Event Register (PM CER)

Offset 0x4

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PMCI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 29-2. PMC Event Register (PM CER)

Table 29-3. PMCER Field Descriptions

Field	Description
PMCI	PMC interrupt Read 1 PMC interrupt event condition occurred. This bit is asserted by PMC when the module is in nap or sleep mode and another CSB master (for example PCI) initiates CSB transactions without sending interrupt to the core in advance. 0 PMC interrupt event did not occur. Sticky bit. Set on listed condition, cleared by writing 1 to this bit. Writing 0 has no effect.

29.2.0.3 PMC Mask Register (PMC MR)

Offset 0x8

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PMCIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 29-3. PMC Mask Register (PMC MR)

Table 29-4. PMC MR Field Descriptions

Field	Description
PMCIE	PMC interrupt Enable 1 Enable interrupt to CPU 0 Disable interrupt to CPU

29.2.0.4 PMC Shadow Register (PMCSR)

Offset 0x0C

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	PMCSR						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 29-4. PMC Shadow Register (PMCSR)

Table 29-5. PMCSR Field Descriptions

Field	Description
PMCSR	Shadow Register. Holds the new CORE_PLL configuration to be used in the core PLL on-the-fly change mode.

29.2.1 Functional description

Based on the four power modes of the embedded low-power e300 Power Architecture CPU core, the module provides five power modes: full-power, doze, nap, sleep, and deep sleep mode. Special sleep modes are provided to allow core PLL reprogramming and pre-divider ratio adjustment. They are called core PLL change mode and pre-divider copy mode. A reference to what is running and what is disabled in the various sleep modes is given in [Table 29-6](#). A reference on how sleep modes are entered and exited is provided in [Table 29-7](#).

Table 29-6. MPC5121e Power Modes

Mode	e300 state	periphery clocks	e300 bus snooping	e300 time base registers	PLL, osc state	DRAM state	Wake-up time
Full Power	Running	Running	Active	Active	Active	Active	< 0.1uS
Doze	Suspended	Running	Active	Active	Active	Active	< 0.1uS

Table 29-6. MPC5121e Power Modes (continued)

Mode	e300 state	periphery clocks	e300 bus snooping	e300 time base registers	PLL, osc state	DRAM state	Wake-up time
Nap	Suspended	Running	Suspended	Active	Active	Depends ¹⁵	< 0.1 μ S ²
Sleep	Suspended	Running	Suspended	Suspended	Active	Depends ¹⁵	< 0.1 μ S ²
Core PII Change	Suspended	Running	Suspended	Suspended	Active	Active	40.000 IPS clk cycles
PRE_DIV Copy	Suspended	Running	Suspended	Suspended	Active	Active ³	16.600 IPS clk cycles
Deep Sleep	Suspended	Suspended	Suspended	Suspended	Power-down ⁴	Self-refresh ⁵	53.300 OSC clk cycles+ 40.000 IPS clk cycle + DRAM wakeup time

¹ DRAM state during nap and sleep mode depends on setting of DDROFF bit in the PMC configuration register, described in 29.2.0.1. When putting DRAM interface in self-refresh mode during nap or sleep, no periphery-generated DRAM access is possible any more, and DRAM transfer initiated by peripherals with master capability (VIU, DIU, SATA, USB, DMA, PCI, FEC, and AXE) is unable to take place. If DRAM is left active, the transfer can continue while the core is in sleep mode.

² To be increased with DRAM wakeup time if DRAM has been configured to self-refresh during nap or sleep mode.

³ DRAM remains active. However, in case DDR1 or DDR2 memory is used, software must generate DRAM PLL reset.

⁴ Other analog blocks, like USB PHY, SATA PHY, USB oscillator are not affected by deep sleep mode. If desired, they need to be put in a power-down state by software prior to entering deep sleep mode.

⁵ DRAM only goes to refresh state when core enters deep sleep mode if DRAM controller has been configured correctly. Please refer to chapter on DRAM controller.

Table 29-7. MPC5121e Power Modes – Sleep and Wake-Up Triggers

Power mode	Events leading to entering this sleep mode	Events leading to return to full power
Doze	Set e300 DOZE bit (HID0[8] ¹ = 1) while - POW bit in MSR ² register is set	Interrupt to the e300 PowerArchitecture core Decrementer interrupt Any reset Machine check exception
Nap with DRAM running	Set e300 NAP bit (HID0[9] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 00001	Interrupt to the e300 PowerArchitecture core Decrementer interrupt Any reset Machine check exception
Nap with DRAM in self-refresh	Set e300 NAP bit (HID0[9] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 00011	Interrupt to the e300 PowerArchitecture core Decrementer interrupt Any reset Machine check exception
sleep with DRAM running	Set e300 SLEEP bit (HID0[10] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 00001	Interrupt to the e300 PowerArchitecture core Any reset Machine check exception
sleep with DRAM in self-refresh	Set e300 SLEEP bit (HID0[10] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 00011	Interrupt to the e300 PowerArchitecture core Any reset Machine check exception
deep sleep	Set e300 SLEEP bit (HID0[10] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 00100	Asynchronous interrupt from GPIO, RTC, CAN Power-on reset

Table 29-7. MPC5121e Power Modes – Sleep and Wake-Up Triggers (continued)

Power mode	Events leading to entering this sleep mode	Events leading to return to full power
Core PLL change mode	Set e300 SLEEP bit (HID0[10] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 01000	Interrupt to e300 PowerArchitecture core Reset Machine check exception
PRE_DIV copy mode	Set e300 SLEEP bit (HID0[10] ¹ = 1) while - POW bit in MSR ² register is set - PMCCR ³ = 10000	Interrupt to e300 PowerArchitecture core Reset Machine check exception

¹ HID0 is a register inside the e300 PowerArchitecture core

² MSR is a register inside the e300 PowerArchitecture core

³ PMCCR register is described in section [29.2.0.1/29-2](#).

NOTE

When the PowerArchitecture core is in debug mode, the PMCCR register should be programmed to 00000. If the register is programmed to any other value, unpredictable operation can result when the PowerArchitecture core hits a breakpoint.

29.2.1.1 Full-Power Mode

This is the default power state of the e300 CPU core. In this power mode, the core is fully powered and the internal functional units are operating at the full-processor clock speed. If the dynamic power management mode is enabled, functional units that are idle automatically enter a low-power state without affecting performance, software execution, or external hardware.

29.2.1.2 Doze Mode

In doze mode, all functional units of the core are disabled except for the time base/decrementer registers and the bus snooping logic.

This mode is entered by programming the doze bit (HID0[8] = 1) when MSR[POW] bit is set. The e300 core enters doze mode several processor clocks after these 2 bits are set.

An asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check input (*mcp*) brings the core into the full-power state. The core in doze mode maintains the phase-locked loop (e300 PLL) in a full-power state and locked to the core external clock input (CSB_CLK), so a transition to the full-power state takes only a few processor clock cycles.

The doze mode is an e300 core only low-power mode, the system stays in full-powered mode while the core enters this mode. The core enters and exits doze mode without being controlled by the PMC.

29.2.1.3 Nap Mode

The nap mode further reduces e300 power consumption by disabling bus snooping, leaving only the time base register and the core PLL in a powered state. While the core enters nap mode, the system stays in full-powered mode.

In this mode, bus snooping is disabled. The core waits until the snoop bus is idle before entering nap mode.

When the DDROFF bit is set in the PMCCR register, described in [29.2.0.1/29-2](#) is set, the DRAM is put into self-refresh mode during nap mode. Setting the DDROFF bit requires the DRAM controller to be configured for putting the DRAM in self-refresh mode upon receipt of the request.

When putting the DRAM in self-refresh mode during nap mode, alternate masters (VIU, DIU, FEC, USB, DMA, AXE, SATA) cannot access DRAM any more. If the DRAM does not enter self-refresh mode, alternate masters can continue reading and writing from DRAM while the core is in a sleep mode.

To enter nap mode, the POW bit in the e300 MSR register must be set, then the COREFOFF bit in the PMCCR register, described in [29.2.0.1/29-2](#) must be set, before setting the nap bit (HID0[9] = 1) in an e300 system register.

The core returns to the full-power state upon receipt of an asynchronous interrupt, system management interrupt, decremter interrupt, hard or soft reset, or machine check input ($\overline{\text{mcp}}$) signal. A return to full-power state from the nap mode takes only a few processor clock cycles if the DRAM is not put in self-refresh mode. If the DRAM is put in self-refresh mode, the time is dominated by the DRAM wakeup time. When the core is in nap mode, another CSB master (for example PCI) may initiate CSB transactions directly without sending an interrupt to the core in advance. A PMC interrupt can be used to wakeup the core if enabled.

29.2.1.4 Sleep Mode

Sleep mode is an e300 core-only sleep mode. In this mode, the e300 enters sleep mode while the system stays in full-powered mode. The e300 sleep mode reduces the core power consumption to a minimum by disabling all core internal functional units.

In this mode, bus snooping is disabled. The core waits until the snoop bus is idle before entering sleep mode.

When the DDROFF bit in the PMCCR register, described in [29.2.0.1/29-2](#) is set, the DRAM is put into self-refresh mode during sleep mode. Setting the DDROFF bit requires the DRAM controller to be configured for putting the DRAM in self-refresh mode upon receipt of the request.

When putting the DRAM in self-refresh mode during sleep mode, alternate masters (VIU, DIU, FEC, USB, DMA, AXE, SATA) cannot access DRAM any more. If the DRAM does not enter self-refresh mode, alternate masters can continue reading and writing from DRAM, while the core is in a sleep mode.

To enter sleep mode, the POW bit in the e300 MSR register must be set, then the COREFOFF bit in the PMCCR register, described in [29.2.0.1/29-2](#) must be set, before setting the sleep bit (HID0[10] = 1) in an e300 system register.

In this mode, the system PLL, core external input clock (CSB_CLK), and e300 PLL are all maintained. The core returns to the full-power state upon receipt of an asynchronous interrupt, system management interrupt, hard or soft reset, or machine check input ($\overline{\text{mcp}}$) signal. A return to full-power state from the sleep mode takes only a few processor clock cycles if the DRAM is not put in self-refresh mode. If the DRAM is put in self-refresh mode, the time is dominated by the DRAM wakeup time. When the core is in

sleep mode, another CSB master (for example PCI) may initiate CSB transfers directly without sending an interrupt to the core in advance. A PMC interrupt can be used to wakeup the core if enabled.

29.2.1.5 Deep Sleep Mode

The system provides a low-power consumption mode where the e300 core enters sleep mode. The system oscillator, system PLL, and e300 PLL are all powered down and disabled. While the module enters this mode, all internal functional units, except the real-time clock (RTC), are disabled. As the clocks are static, the current draw of the device is reduced to leakage level. The internal state of the device is maintained in deep sleep as long as power is maintained. The real-time clock (RTC) is not disabled in deep sleep mode. If the RTC is used, that portion of the chip continues consuming power in deep sleep mode.

In this mode, bus snooping is disabled. The core waits until the snoop bus is idle before entering deep sleep mode.

The DRAM is put into self-refresh mode during deep sleep mode. It is required that the DRAM controller is configured for putting the DRAM in self-refresh mode upon receipt of the request.

To enter deep sleep mode, the POW bit in the e300 MSR register must be set, then the DSM bit in the PMCCR register, described in [29.2.0.1/29-2](#) must be set, before setting the sleep bit (HID0[10] = 1) in an e300 system register

Any hard reset, or an asynchronous interrupt from GPIO, RTC or one of the CAN modules (which occurs when a data transition occurs on the serial input) can be used to bring the module back to the full-power state from deep sleep mode. No clock is required to trigger the wake up process in the case of the GPIO interrupt or the CAN module interrupt. However, the RTC clock must be present and running to wake up the core from deep sleep. On reception of a wakeup signal from GPIO, RTC, or MSCAN, PMC enables the system oscillator and system PLL in turn and waits for the oscillator to stable and system PLL and e300 PLL to lock one-by-one. After the system clocks are back to full-power mode, the interrupt from GPIO, RTC, MSCAN, or any other peripheral may bring the e300 core back to the full-power state. The wakeup time is listed in [Table 29-6](#).

During deep sleep mode, state of all peripherals is frozen. All state and data is retained in internal registers, but the peripherals stop reacting on triggers from outside (e.g. a UART does not see incoming data any more). Reinitialization may be needed after leaving deep sleep mode.

- Reinitialization is not needed if the peripheral does not need to resynchronize to an external interface, or if the resynchronization can be done without reinitialization
- Reinitialization is needed if the peripheral requires it to sync again with an external interface.

Reset is not needed after deep sleep mode.

NOTE

To use the RTC to wakeup module from the deep sleep mode, the ATC timeout and periodic interrupt (or alarm/stopwatch interrupt) must be used together. Program the RTC target time (TTR) register and enable one of the interrupt before entering the e300 to sleep mode.

NOTE

While a pulse is applied on a GPIO pin to wakeup the module from deep sleep mode, the pulse should be wider than 2 ms.

29.2.2 Core PLL Change Mode

This mode can change the core PLL setting on-the-fly. To change core PLL setting, write the new setting to the shadow register (PMCSR) in PMC module, and then set the CCM bit in register PMCCR described in section 29.2.0.1/29-2. Because the core PLL needs relock, the core must enter sleep mode.

To enter core PLL change mode, the POW bit in the e300 MSR register must be set, then the CCM bit in the PMCCR register, described in 29.2.0.1/29-2 must be set, before setting the sleep bit (HID0[10] = 1) in an e300 system register

In this mode, bus snooping is disabled. The core waits until the snoop bus is idle before entering Core PLL change mode.

After the core enters sleep mode, PMC provides the updated core PLL configuration to the core.

During the core PLL change process, PMC blocks all interrupts to the core to make sure it's not waked up while the PLL is not locked. The wakeup time is listed in Table 29-6. After the wake time, the PMC releases the interrupts and the PMC interrupt or any other interrupt can be used to wakeup the core from sleep mode.

29.2.3 PRE_DIV Copy Enable Mode

This mode can change the system clock divide ratio (SYS_DIV), also referred to as pre-divider ratio (PRE_DIV). To change the system clock divide ratio, write the new SYS_DIV value to the shadow register (SCFR2) in the CLOCK module, and then set the PRE_DIV bit in register PMCCR described in section 29.2.0.1/29-2. Because the core PLL needs relock, the core must enter sleep mode. To enter sleep mode, the POW bit in the e300 MSR register must be set, then set the sleep bit (HID0[10] = 1).

NOTE

If DDR1 or DDR2 DRAM is used, the DRAM PLL needs to relock because DRAM clock changes frequency. This needs to be done in software.

NOTE

Changing pre-divider factor changes all on-chip frequencies. This may upset certain peripherals, and require re-initialization. The device drivers of these peripherals must take care of this.

In this mode, bus snooping is disabled. The core waits until the snoop bus is idle before entering Core PLL change mode.

After the core enters sleep mode, PMC waits several processor clocks for the core PLL to be switched off, and then asserts a copy_shadow signal to enable the CLOCK module to copy the new SYS_DIV value and update system clock frequency.

During the PRE_DIV copy enable process (i.e. from the point when the core asserts the $\overline{\text{qreq}}$ signal to when the core PLL is locked to the new system clock input), PMC blocks all interrupts to the core to make sure it's not waked up while the PLL is not locked. The wakeup time is listed in [Table 29-6](#). After the wakeup time, the PMC releases the interrupts and the PMC interrupt or any other interrupt can be used to wakeup the core from sleep mode. Then, the system clock divide ratio change is done.

29.2.4 Low-Power Configurations

The following table summarizes the valid PMC and e300 settings that can be used to put the module into low-power mode, core PLL on-the-fly change mode, or pre-divider ratio copy enable mode.

Table 29-8. Valid PMC/e300 Low-Power Configurations

Mode to Enter	e300 Mode to Enter	PRE_DIV PMCCR[4]	CCM PMCCR[3]	DSM PMCCR[2]	DDROFF PMCCR[1]	COREOFF PMCCR[0]	e300 MSR[POW]	e300 HID0[8:10]
Doze	Doze	x	x	x	x	x	1	100
Nap	Nap	0	0	0	0	1	1	010
Nap	Nap	0	0	0	1	1	1	010
Sleep	Sleep	0	0	0	0	1	1	001
Sleep	Sleep	0	0	0	1	1	1	001
Deep Sleep	Sleep	0	0	1	0	0	1	001
Core PLL Change	Sleep	0	1	0	0	0	1	001
PRE_DIV Copy	Sleep	1	0	0	0	8	1	001

Chapter 30

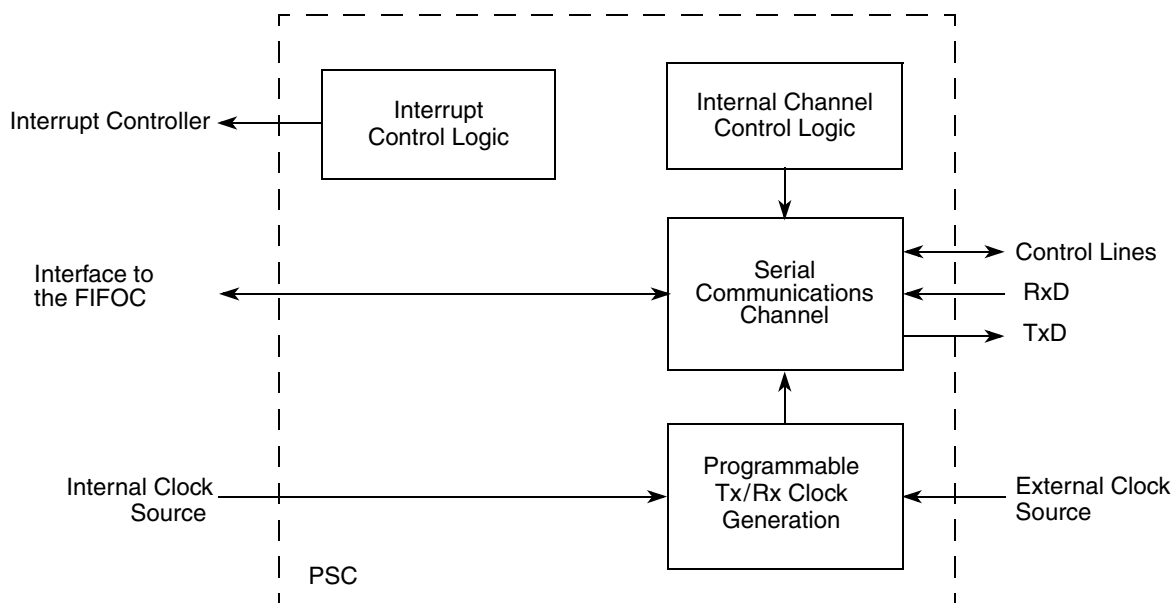
Programmable Serial Controller (PSC)

30.1 Introduction

Each PSC can be clocked by an internal or external clock source. [Figure 30-1](#) shows a simplified PSC block diagram. In addition, each PSC module interfaces directly to the CPU and consists of the following:

- Serial communication channel
- Programmable transmit (Tx) receive (Rx) clock generation
- Internal channel control logic
- Interrupt control logic

The PSC also provides a MCLK for the external codec, eliminating the need for an external crystal for the external device. For more information about the codec mode, see [Section 30.5.2, “PSC in Codec Mode”](#).



Note: During Codec Mode the number of available TxD and RxD lines is depend on the configuration of the Rx-Tx Channels fields of register MR2.

Figure 30-1. Block Diagram

30.2 Memory Map

The register address is calculated as base address for the regarding PSC plus the offset value. [Table 30-1](#) shows the list with all implemented registers and the associated offset value.

Table 30-1. PSC Memory Map

Offset	Register Name	Register Width	Access
00	Mode Register 1 (MR1)	8	R/W
00	Mode Register 2 (MR2)	8	R/W
04	Status Register (SR)	16	R
04	Clock Select Register (CSR)	16	W
08	Command Register (CR)	8	W
0C	Rx Buffer Register (RB)	32	R
0C	Tx Buffer Register (TB)	32	W
10	Input Port Change Register (IPCR)	8	R
10	Auxiliary Control Register (ACR)	8	W
14	Interrupt Status Register (ISR)	16	R
14	Interrupt Mask Register (IMR)	16	W
18	Counter Timer Upper Register (CTUR)	8	W
1C	Counter Timer Lower Register (CTLR)	8	W
20	Codec Clock Register (CCR)	32	R/W
24	AC97 Slots Register (AC97Slots)	32	W
28	AC97 Command Register (AC97CMD)	32	R/W
2C	AC97 Status Data Register (AC97Data)	32	R
30	Reserved		
34	Input Port Register (IP)	8	R
38	Output Port 1 Bit Set (OP1)	8	W
3C	Output Port 0 Bit Set (OP0)	8	W
40	Serial Interface Control Register (SICR)	32	R/W

PSC module operation is controlled by writing control bytes into the appropriate registers.

30.2.1 Register Descriptions

The terms assertion and negation are used to avoid confusion between active-low and active-high signals. Asserted indicates a signal is active, independent of the voltage level. Negated indicates a signal is inactive.

30.2.1.1 Mode Register 1 (MR1)

The mode registers control configuration. MR1 can be read or written when the mode register pointer points to it, at reset or after a reset mode register pointer command using [CR\[MISC\]](#). After MR1 is read or written, the pointer points to [MR2](#).

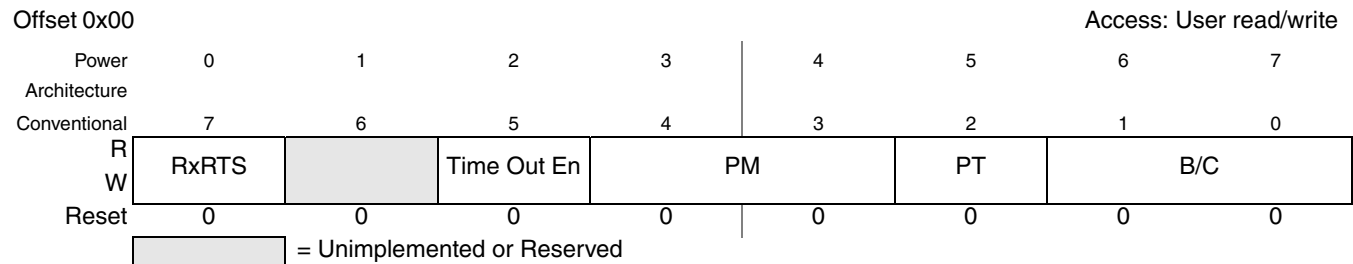


Figure 30-2. Mode Register 1 for UART Mode (MR1)

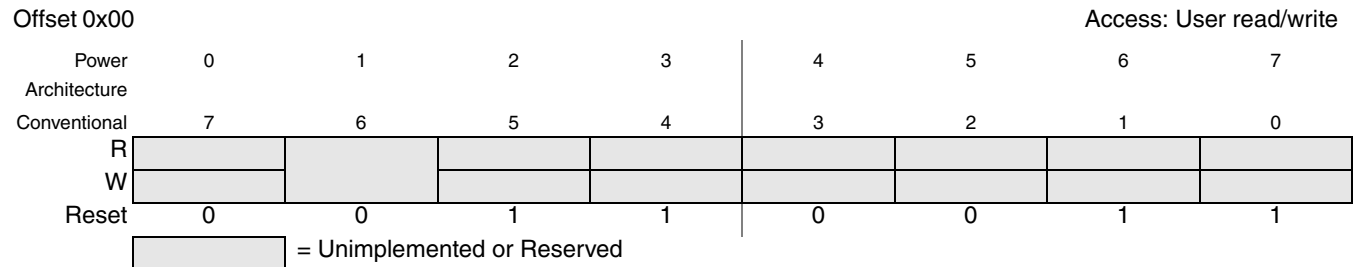


Figure 30-3. Mode Register 1 for Other Modes (MR1)
(Registers are repeated for reference.)

Table 30-2. MR1 Field Descriptions

Field	Description
RxRTS	UART. Receiver request-to-send. Allows RTS output to control the CTS input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for RTS control, RTS control is disabled for both. Transmitter RTS control is configured in MR2[TxRTS]. Not used in codec mode. 0 Receiver has no effect on $\overline{\text{RTS}}$. 1 When a valid start bit is received, $\overline{\text{RTS}}$ is negated if the PSC FIFO is full. Other Modes. Reserved.
Time Out En	UART. Enable the time out counter. If the time out counter is disabled, the TIME OUT status in the SR is also cleared. 0 Time out counter is disabled 1 Time out counter is enabled Other Modes. Reserved.
PM	UART. Parity mode. Selects the parity or multi-drop mode for the channel. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. The value of PM affects PT, as shown Table 30-3 . PM is not used in codec mode. Other Modes. Reserved.

Programmable Serial Controller (PSC)

Offset 0x00

Access: User read/write

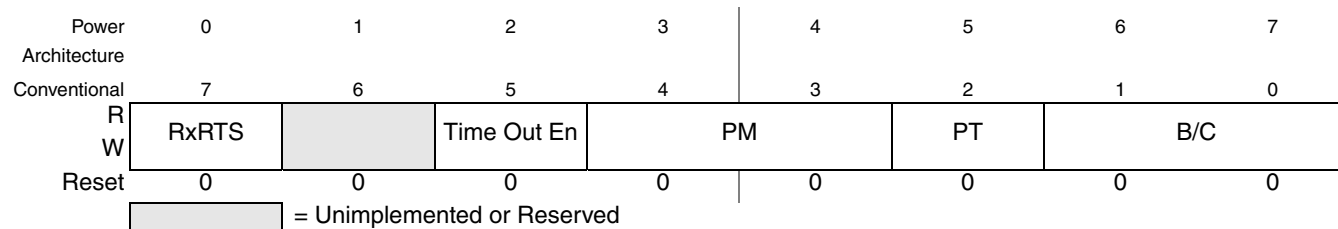


Figure 30-2. Mode Register 1 for UART Mode (MR1)

Offset 0x00

Access: User read/write

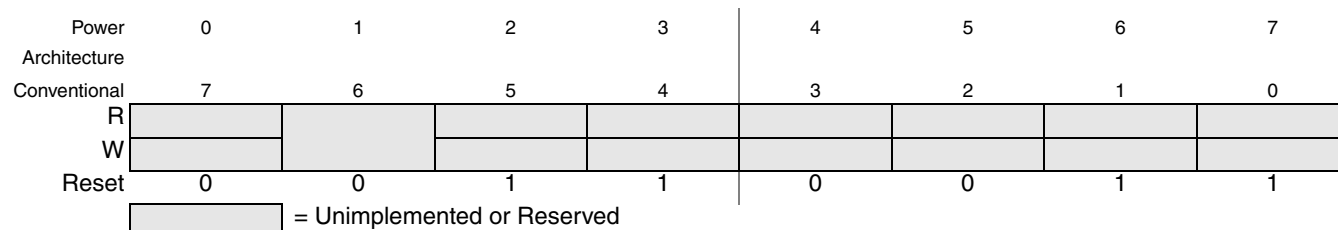


Figure 30-3. Mode Register 1 for Other Modes (MR1)
(Registers are repeated for reference.)

Table 30-2. MR1 Field Descriptions (continued)

Field	Description
PT	UART. Parity Type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11). PT is not used in codec mode. See Table 30-3 . Other Modes. Reserved.
B/C	UART. Bits per Character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits. B/C is not used in codec mode. 00 5 bits 01 6 bits 10 7 bits 11 8 bits Other Modes. Reserved.

Table 30-3. Parity Mode/Parity Type Definitions

PM	Parity Mode	Parity Type (PT=0)	Parity Type (PT=1)
00	With parity	Even parity	Odd parity
01	Force parity	Low parity	High parity
10	No parity	n/a	
11	Multidrop mode	Data character	Address character

30.2.1.2 Mode Register 2 (MR2)

MR2 can be read or written when the Mode register pointer points to it, which occurs after any access to MR1. An MR2 access does not update the mode register address.

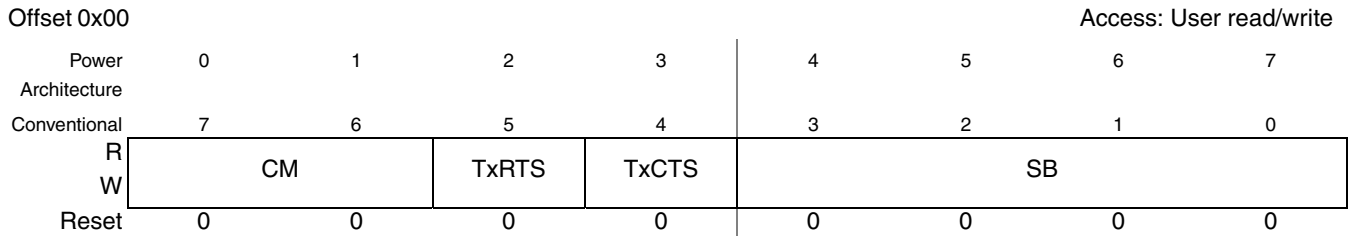


Figure 30-4. Mode Register 2 for UART Mode (MR2)

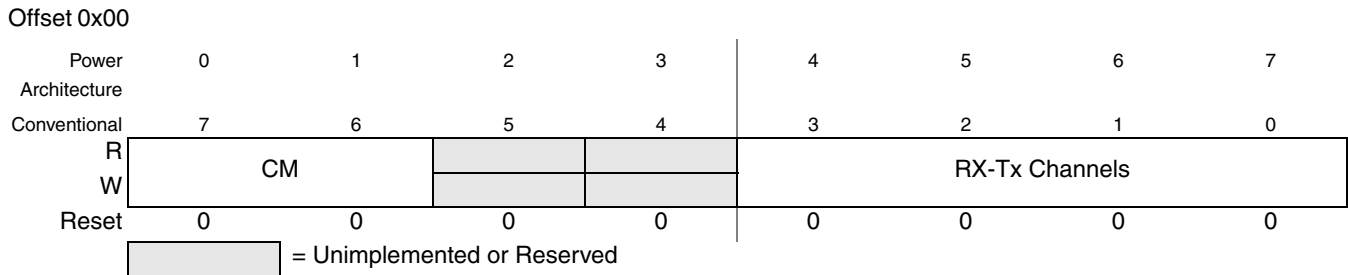


Figure 30-5. Mode Register 2 for Other Modes (MR2)
(Registers are repeated for reference.)

Table 30-4. MR2 Field Descriptions

Field	Description
CM	Channel mode. Selects a channel mode. CM is used in UART and codec modes. 00 Normal 01 Automatic echo 10 Local loop-back 11 Remote loop-back
TxRTS	UART. Transmitter ready-to-send. Controls negation of RTS to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same channel for $\overline{\text{RTS}}$ control is not permitted and disables $\overline{\text{RTS}}$ control for both. TxRTS is not used in codec mode. 0 The transmitter has no effect on $\overline{\text{RTS}}$. 1 Setting this bit automatically clears RTS line one bit-time after any characters in the transmitter shift registers are completely sent, including the programmed number of stop bits. Other Modes. Reserved.
TxCTS	UART. Transmitter clear-to-send. If TxCTS and TxRTS are enabled, TxCTS controls the operation of the transmitter. TxCTS is not used in Codec mode. 0 $\overline{\text{CTS}}$ has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\text{CTS}}$ each time it is ready to send a character. If $\overline{\text{CTS}}$ is asserted then character is sent If it is negated, the channel TxD remains in a high state and transmission is delayed until $\overline{\text{CTS}}$ is asserted. Changes in $\overline{\text{CTS}}$ as a character is being sent do not affect its transmission. Other Modes. Reserved.

Offset 0x00

Access: User read/write

Power	0	1	2	3	4	5	6	7
Architecture								
Conventional	7	6	5	4	3	2	1	0
R	CM		TxRTS		TxCTS		SB	
W								
Reset	0	0	0	0	0	0	0	0

Figure 30-4. Mode Register 2 for UART Mode (MR2)

Offset 0x00

Power	0	1	2	3	4	5	6	7
Architecture								
Conventional	7	6	5	4	3	2	1	0
R	CM				RX-Tx Channels			
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 30-5. Mode Register 2 for Other Modes (MR2)
(Registers are repeated for reference.)**
Table 30-4. MR2 Field Descriptions (continued)

Field	Description
SB	UART. Stop-Bit (length control). Selects the stop bit length appended to the transmitted character. Stop-bit lengths of 9/16th to 2 bits are programmable for 6- and 8-bit characters. Lengths of 1 1/16th to 2 bits are programmable for 5-bit characters. In all cases, the receiver checks only for a high condition at the center of the first stop-bit position, that is, one bit-time after the last data bit or after the parity bit, if parity is enabled. Therefore, the receiver doesn't support a stop bit length less than one. Not used in codec mode, see Table 30-5 . Other Modes. Reserved.
RX - TX Channels	Codec. Define the number of used Rx and Tx channels. Up to three Rx and Tx channels are supported, but the PSC interface only supports three data lines (Rx + Tx). Therefore, the maximum number of Rx channels is also depend on the programmed number of Tx channels. 0000 = 1 Rx, 1 Tx 0001 = 1 Rx, 2Tx 0010 = 0 Rx, 3Tx 0100 = 2 Rx, 1 Tx 1000 = 3 Rx, 0Tx other = not supported For more information about the pin muxing see Table 30-25 Other Modes. Reserved.

Table 30-5. Stop-Bit Lengths

SB	5 Bits	6–8 Bits	SB	5 Bits	6–8 Bits	SB	5–8 Bits	SB	5–8 Bits
0000	1.063	0.563	0100	1.313	0.813	1000	1.563	1100	1.813
0001	1.125	0.625	0101	1.375	0.875	1001	1.625	1101	1.875
0010	1.188	0.688	0110	1.438	0.938	1010	1.688	1110	1.938
0011	1.250	0.750	0111	1.500	1.000	1011	1.750	1111	2.000

30.2.1.3 Status Register (SR)

The read-only SR register shows status of the transmitter, the receiver, and the FIFO.

Offset 0x04

Access: User read only

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RB/EOF	FE	PE	ORERR	TXEMP				CDE	ERROR	TIMEOUT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= UNIMPLEMENTED OR RESERVED

Figure 30-6. Status Register for UART Mode (SR)

Offset 0x04

Access: User read only

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				ORERR	URRR					ERROR			CMDSEN D	DATA_OVR	DATA_VALID	UNEX_RX_SLOT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-7. Status Register for Other Mode (SR)
(Registers are repeated for reference.)

Table 30-6. SR Field Descriptions (Sheet 1 of 3)

Field	Description
RB/EOF	UART. Received Break. Detects breaks originating in middle of received character. Such a break must persist until the end of next detected character time. 0 No break received. 1 An all-0 character of the programmed length was received without a stop bit. Only a single FIFO position is occupied when a break is received. Further entries to FIFO are inhibited until Rx D returns to high state for at least one-half bit-time, which equals two successive PSC clock edges. Other Modes. Reserved.
FE	UART. Framing Error. Not used (always 0) in codec mode. 0 No framing error occurred. 1 No stop bit detected when corresponding FIFO data character received. Stop bit-check occurs in middle of first stop bit position. Other Modes. Reserved.
PE	UART. Parity Error. PE is not used (always 0) in codec mode. 0 No parity error occurred. 1 If MR1[PM] equals 0x (with parity or force parity), corresponding FIFO character was received with incorrect parity. If MR1[PM] equals 11 (multidrop), PE stores received A/D bit. Other Modes. Reserved.
ORERR	Overrun Error. Indicates whether an overrun occurs. For purposes of overrun, FIFO full means all FIFO space is occupied; the Rx FIFO threshold is irrelevant to overrun. 0 No overrun occurred. 1 One or more characters in Rx data stream were lost. ORERR sets on receipt of a new character when FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the Rx shift register and its break detect, framing error status, and parity error, if any, are lost. This bit is cleared by a reset error status command in the CR and not by a reset Tx command in the CR.

Programmable Serial Controller (PSC)

Offset 0x04

Access: User read only

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RB/EOF	FE	PE	ORERR	TXEMP				CDE	ERROR	TIMEOUT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= UNIMPLEMENTED OR RESERVED

Figure 30-6. Status Register for UART Mode (SR)

Offset 0x04

Access: User read only

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				ORERR	URERR					ERROR			CMD_SEN D	DATA_OVR	DATA_VALI D	UNEX_RX _SLOT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 30-7. Status Register for Other Mode (SR)
(Registers are repeated for reference.)**

Table 30-6. SR Field Descriptions (Sheet 2 of 3)

Field	Description
TXEMP/URERR	UART. Transmitter Empty. 0 Tx buffer not completely empty. Either a character is being shifted out, or Tx is disabled. Tx is enabled/disabled by programming CR [TC]. 1 Tx has underrun (both the Tx holding register and Tx shift registers are empty). This bit sets after transmission of the last stop bit of a character, if there are no characters in the Tx holding register awaiting transmission. Other Modes. Underrun Error. 0 No error. 1 Underrun error occurred, which means the number of Tx FIFO bytes is 0, the Tx shift register is empty, and a FrameSync occurs. In other words, the time has come to transmit a new sample, but no sample is available in the Tx shift register. Unlike UART mode, TxEMP high indicates an error condition similar to the overrun condition (ORERR = 1). It is now cleared the same way as ORERR by a RESET ERROR STATUS command in the CR and not by a reset Tx command in the CR.
CDE/DEOF	UART. DCD Status. 0 The $\overline{\text{DCD}}$ input is negated while receiving data. 1 No error Other Modes. Reserved
ERROR	Error Status Detect. 0 No errors connected 1 The PSC controller detect an error state. This error is a combination of the error bits: RB, FE, PE, URERR, ORERR from this register and RX and RX FIFO bit from the TFSTAT and RFSTAT register. Other Modes. Reserved.
TIME OUT	UART. Time Out. 0 No Time out event occurred 1 Time out event occurred. Other Modes. Reserved

Offset 0x04

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RB/EOF	FE	PE	ORE RR	TXE MP				CDE	ERR OR	TIME OUT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= UNIMPLEMENTED OR RESERVED

Figure 30-6. Status Register for UART Mode (SR)

Offset 0x04

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R				ORE RR	URE RR					ERR OR			CMD _SEN D	DATA _OVR	DATA _VALI D	UNE X_RX _SLOT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

 Figure 30-7. Status Register for Other Mode (SR)
(Registers are repeated for reference.)

Table 30-6. SR Field Descriptions (Sheet 3 of 3)

Field	Description
CMD_SEND	AC97 Mode. Command Send Ready. 0 The data in the AC97CMD register was sent out by the AC97 transmitter 1 The data in the AC97CMD register was not sent out. A write access to the AC97CMD register sets this bit to one. After the AC97 transmitter sends out the CMD data, this bit is cleared. Other Modes. Reserved
DATA_OVR	AC97 Mode. Receive Status Data Overwrite. 0 No received status data overwrite. 1 The received frame contains a new valid data status word in slot 2, but the previous received status data word was not read out before the new one was written to the AC97data register. Therefore, the old status data word was lost. A read access to the AC97 data register clears this bit. Other Modes. Reserved.
DATA_VALID	AC97 Mode. Received Status Data. 0 The received frame doesn't contains valid status data. 1 The received frame contains a valid data status word in slot 2. The received data are located in the AC97 data register. A read access to the AC97data register clears this bit. Other Modes. Reserved.
UNEX_RX_SLOT	AC97 Mode. Unexpected Receive Slots Detect. 0 The received frame contains the slots defined in the AC97 slots register or a frame without AC97 data (frame is empty or contains only slot 1 or slot 2 data). The case that the receive frame contains all expected data slots plus additional data slots is also excepted. The Rx data of the frame that matches this rule is written to the RX FIFO. 1 The AC97 receiver detects a frame which is not matching the rules above. The Rx data of the frame which triggers this bit is ignored. Other Modes. Reserved.

30.2.1.4 Clock Select Register (CSR)

The device supports internal and external clocks as source for the UART clock generation. For the UART clock generation, a prescaler by 32 or 10 is available. After reset, the prescaler by 10 for the UART mode is selected.

Offset 0x04

Access: User write only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	RCS				TCS											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-8. Clock Select Register for UART Mode (CSR)

Offset 0x04

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-9. Clock Select Register for Other Modes (CSR)

Table 30-7. CSR Field Descriptions

Field	Description
RCS	UART. Receiver Clock Select Register 0000–1101 Choose the prescaler by 32 for the UART receive clock generation 1110 Choose the external clock source 1111 Choose the prescaler by 10 for the UART receive clock generation Other Modes. Reserved
TCS	UART. Transmitter Clock Select Register 0000–1101 Choose the prescaler by 32 for the UART transmit clock generation 1110 Choose the external clock source 1111 Choose the prescaler by 10 for the UART transmit clock generation Other Modes. Reserved

30.2.1.5 Command Register (CR)

The command registers (CR) provide the commands to the PSC in all modes. Only multiple commands that do not conflict can be specified in a single write to a CR. For example, reset Tx and enable Tx cannot be specified in one command.

Offset 0x08

Access: User read/write



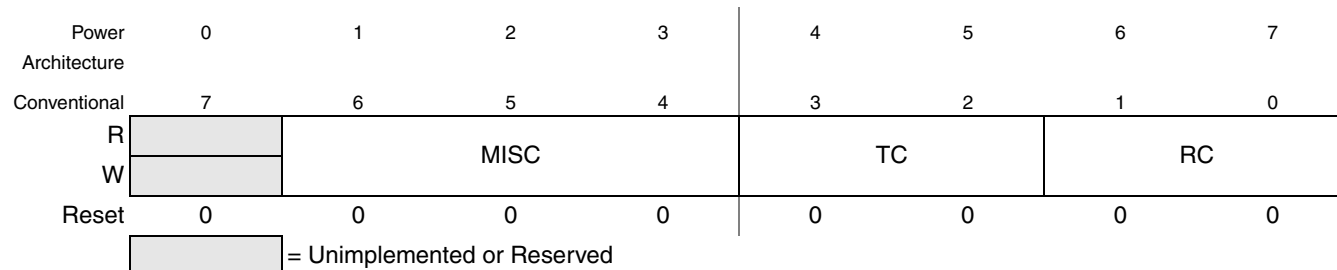
Figure 30-10. Command Register for all Modes (CR)
(Register is repeated for reference.)

Table 30-8. CR Field Descriptions (Sheet 1 of 3)

Field	Value	Command	Description
MISC	000	No command	—
	001	Reset mode register pointer	Causes MR register address to point to MR1.
	010	Reset receiver	Immediately disables receiver and re-initializes receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfigure the receiver.
	011	Reset transmitter	In UART mode, this bit immediately disables Tx and clears SR[TxEMP]. No other registers are altered. Because it places Tx in a known state, use this command instead of transmitter disable when reconfigure transmitter. In codec mode, the Tx prefetch register is cleared and URERR is not cleared by this soft reset. It is cleared the same way as the Rx overflow bit, by a RESET ERROR STATUS command.
	100	Reset error status	In UART mode, clears ISR[RB, FE, PE, ORERR]. In codec mode, command clears ISR[ORERR, URERR, DEOF, CMD_SEND, DATA_OVR, DATA_VALID, and UNEX_RX_SLOT]
	101	Reset break change interrupt	Clears the delta break bit, ISR[DB]. Command has no effect in codec mode.
	110	Start break	Forces TxD low <ul style="list-style-type: none"> If Tx is empty, break may be delayed up to one bit-time. If Tx is active, break starts when character transmission completes. Break is delayed until any character in Tx shift register is sent. Any character in Tx holding register is sent after the break. Tx must be enabled for command to be accepted. This command ignores the CTS state and has no effect in codec mode.
	111	Stop break	Causes TxD to go high (mark) within two bit-times. Any characters in the Tx buffer are sent.

Offset 0x08

Access: User read/write



**Figure 30-10. Command Register for all Modes (CR)
(Register is repeated for reference.)**

Table 30-8. CR Field Descriptions (Sheet 2 of 3)

Field	Value	Command	Description
TC	00	No action taken	Causes Tx to stay in current mode. <ul style="list-style-type: none"> If Tx is enabled, it remains enabled. If Tx is disabled, it remains disabled.
	01	Transmitter enable	Enables operation of Tx channels. SR[TxEMP] sets. If Tx is already enabled, this command has no effect. In UART Mode: TxEMP bits in SR become asserted. In Codec Mode: Tx FIFO can be loaded while Tx is disabled, unlike in UART mode. Therefore this command does not affect URERR behavior. It does not automatically set URERR. If no data is written to Tx FIFO, URERR sets at the first FrameSync after Tx is enabled. In AC97 Mode: URERR sets if Tx FIFO is empty, Tx is enabled, Rx detects a codec ready condition, and a FrameSync occurs before samples are written to the Tx FIFO. Note: In codec/AC97 mode, it's not possible to use the transmitter without the receiver. To transmit data only, the receiver must be enabled.
	10	Transmitter disable	Terminates Tx operation and clears SR[TxEMP]. <ul style="list-style-type: none"> If a character is being sent when Tx is disabled, transmission completes before Tx becomes inactive. If Tx is already disabled, the command has no effect. In UART Mode: SR[TxEMP] are negated. In Codec Mode: SR[TxEMP] is negated. Tx does not clear unless PSC is in remote loop-back or auto-echo mode. In codec mode, unlike UART mode, the Tx FIFO may be loaded while Tx is disabled.
	11	—	Reserved, do not use

Offset 0x08

Access: User read/write



Figure 30-10. Command Register for all Modes (CR)
(Register is repeated for reference.)

Table 30-8. CR Field Descriptions (Sheet 3 of 3)

Field	Value	Command	Description
RC	00	No action taken	Causes receiver to stay in current mode. <ul style="list-style-type: none"> If receiver is enabled, it remains enabled. If receiver is disabled, it remains disabled.
	01	Receiver enable	Enables receiver <ul style="list-style-type: none"> If PSC module is not in multidrop mode ($MR1[PM] \neq 11$), RECEIVER ENABLE command enables channel's receiver and forces it into a search-for-start-bit state. In multidrop mode the Rx continuously monitors the received data regardless of whether it is enabled or not. If receiver is already enabled, this command has no effect.
	10	Receiver disable	Immediately disables receiver. In UART mode any character being received is lost. The command does not affect receiver status bits or other control registers. <ul style="list-style-type: none"> If the PSC module is programmed for local loop-back or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect. In codec mode, if the receiver is disabled while a character is being received, reception completes before the receiver becomes inactive.
	11	—	Reserved, do not use.

Note: The RC field selects a single command.

30.2.1.6 Rx Buffer Register (RB)

Read access from this read only register reads the data directly from the Rx shift register. This access bypasses the data in the RX FIFO. Use the data register in the RX FIFO to read the Rx data.

Offset 0x0C												Access: User read only				
Power Architecture	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RB[0:15]															
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RB[16:31]															
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 30-11. Rx Buffer Register for UART8/16/32 Modes (RB)

Offset 0x0C												Access: User read only				
Power Architecture	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RB[0:15]															
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RB[16:19]				SOF											
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 30-12. Rx Buffer Register for AC97 Mode (RB)

Offset 0x0C

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RB[0:15]															
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RB[16:23]															
W	Used by Tx Buffer															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-13. Rx Buffer Register for Codec24 Mode (RB)

Table 30-9. RB Field Descriptions

Field	Description
RB	<p>AC97 (0:19). Received data. AC97 data must be read one complete sample at a time, where all samples except time slot #0 are 20 bits. Time slot #0 data is in bits 0:15. Bit 20 is 1 in the first sample of a new frame. Bit 20 contains the start of frame indicator:</p> <p>0 RB[0:19] is not the first sample in the frame.</p> <p>1 RB[0:15] is the first sample in a new frame. The number 0 slot is called the TAG slot.</p> <p>The bits [21:31] are reserved at this mode.</p> <p>UART/Codec8 (0:31). Received data. For these modes, data can be read one, two, or four bytes at a time. For one byte at a time, all bytes must be read from bits 0:7. For two bytes at a time, data must be read from bits 0:15. Lower-bit data was received before upper-bit data.</p> <p>Codec16 (0:31). Received data. For these modes, data can be read two or four bytes at a time. For two bytes at a time, data must be read from bits 0:15. Lower-bit data was received before upper-bit data.</p> <p>Codec24 (0:23). Received data. For these modes, data must be read four bytes at a time. The lower 24 bits contain the received data word.</p> <p>Codec32 (0:31). Received data. For these modes, data must be read four bytes at a time.</p>

30.2.1.7 Tx Buffer Register (TB)

Writing to this register places data directly into the Tx shift register. This access bypasses the data in the Tx FIFO. Use the data register in the Tx FIFO to provide the Tx data.

Offset 0x0C																Access: User read only			
Power Architecture	msb 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R	Used by Rx Buffer																		
W	TB[0:15]																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
R	Used by Rx Buffer																		
W	TB[16:31]																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

= Unimplemented or Reserved

Figure 30-14. Tx Buffer Register for UART/Codec8/16/32 Modes (TB)

Offset 0x0C																Access: User read only			
Power Architecture	msb 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R	Used by Rx Buffer																		
W	TB[0:15]																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
Power Architecture	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Conventional	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
R	Used by Rx Buffer																		
W	TB[16:19]																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			

= Unimplemented or Reserved

Figure 30-15. Tx Buffer Register for AC97 Mode (TB)

Offset 0x0C

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Used by Rx Buffer																
W	TB[0:15]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Used by Rx Buffer																
W	TB[16:23]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-16. Tx Buffer Register for Codec24 Mode (TB)

Table 30-10. TB Field Descriptions

Field	Description
TB	<p>AC97 (0:19). Transmit data. AC97 data must be written one complete sample at a time, where all samples except time slot #0 are 20 bits. Time slot #0 data is in bits 0:15.</p> <p>0 RB[0:19] is not the first sample in the frame.</p> <p>1 RB[0:15] is the first sample in a new frame. The number 0 slot is called the TAG slot.</p> <p>The bits [21:31] are reserved at this mode.</p> <p>AC97 (0:19). Transmit data. AC97 data for the expected time slots (3 to 12). The lower 20 bits contain the valid data word.</p> <p>UART/Codec8 (0:31). Transmit data. For these modes, data can be written one, two, or four bytes at a time. For one byte at a time, all bytes must be written to bits 0:7. For two bytes at a time, data must be written to bits 0:15. Lower-bit data is stored before upper-bit data.</p> <p>Codec16 (0:31). Transmit data. For these modes, data can be written two or four bytes at a time. For 2 bytes at a time, data must be written to bits 0:15. Lower-bit data is stored before upper-bit data.</p> <p>Codec24 (0:23). Transmit data. For these modes, data must be written four bytes at a time. The lower 24 bits contain the valid data word.</p> <p>Codec32 (0:31). Transmit data. For these modes, data must be written four bytes at a time.</p>

30.2.1.8 Input Port Change Register (IPCR)

The read-only IPCR register shows the current state and change-of-state for the modem control input port.

Offset 0x010

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7 lsb
Conventional		7	6	5	4	3	2	1	0
R				D_DCD	D_CTS			DCD	CTS
W									
Reset		0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 30-17. Input Port Change Register for UART Modes (IPCR)

Offset 0x010

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7 lsb
Conventional		7	6	5	4	3	2	1	0
R		SYNC		D_DCD	D_CTS			DCD	CTS
W									
Reset		0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 30-18. Input Port Change Register for Codec Mode (IPCR)

Table 30-11. IPCR Field Descriptions

Field	Description
SYNC	Codec. Sync detected. 0 Has not detected sync. 1 Detected sync condition (FrameSync signal = 1 in Codec Modes or Sync = 1 in AC97 mode) Other Modes. Reserved. A read access to this register clears the SYNC bit
D_DCD	Delta DCD. 0 No change-of-state has occurred since the last time the CPU read the IPCR. A read of the IPCR also clears the IPCR D_DCD bit. 1 A change of state (1/16 or 1 bit duration determined by the CSR, CTUR and CTLR) has occurred at $\overline{\text{DCD}}$ input. When this bit is set, the ACR can be programmed to generate an interrupt to the processor.
D_CTS	Delta CTS. 0 No change-of-state has occurred since the last time the CPU read the IPCR. A read of the IPCR also clears the IPCR D_CTS bit. 1 A change of state, lasting a certain time, has occurred at $\overline{\text{CTS}}$ input. When this bit is set, the ACR can be programmed to generate an interrupt to the processor. After the enable of the PSC, the CPU must read this bit to make sure this bit is cleared at the beginning of the transmission.
DCD	Current state of $\overline{\text{DCD}}$ port. This input is double latched. 0 The current state of the DCD input port is low. 1 The current state of the DCD input port is high.

Offset 0x010

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7 lsb
Conventional		7	6	5	4	3	2	1	0
R				D_DCD	D_CTS			DCD	CTS
W									
Reset		0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 30-17. Input Port Change Register for UART Modes (IPCR)

Offset 0x010

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7 lsb
Conventional		7	6	5	4	3	2	1	0
R		SYNC		D_DCD	D_CTS			DCD	CTS
W									
Reset		0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 30-18. Input Port Change Register for Codec Mode (IPCR)

Table 30-11. IPCR Field Descriptions (continued)

Field	Description
CTS	Current state of $\overline{\text{CTS}}$ port. This input is double latched. 0 The current state of the $\overline{\text{CTS}}$ input port is low. 1 The current state of the $\overline{\text{CTS}}$ input port is high.

30.2.1.9 Auxiliary Control Register (ACR)

The write-only ACR register controls Tx/Rx handshaking.

Offset 0x010

Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7 lsb
Conventional		7	6	5	4	3	2	1	0
R									
W								IEC1	IEC0
Reset		0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 30-19. Auxiliary Control Register for all Modes (ACR)

Table 30-12. ACR Field Descriptions

Field	Description
IEC1	Interrupt Enable Control for D_DCD. 0 D_DCD has no effect on the IPC in the ISR. 1 When the D_DCD becomes high, IPC bit in the ISR sets (causing an interrupt if mask is set).



Programmable Serial Controller (PSC)

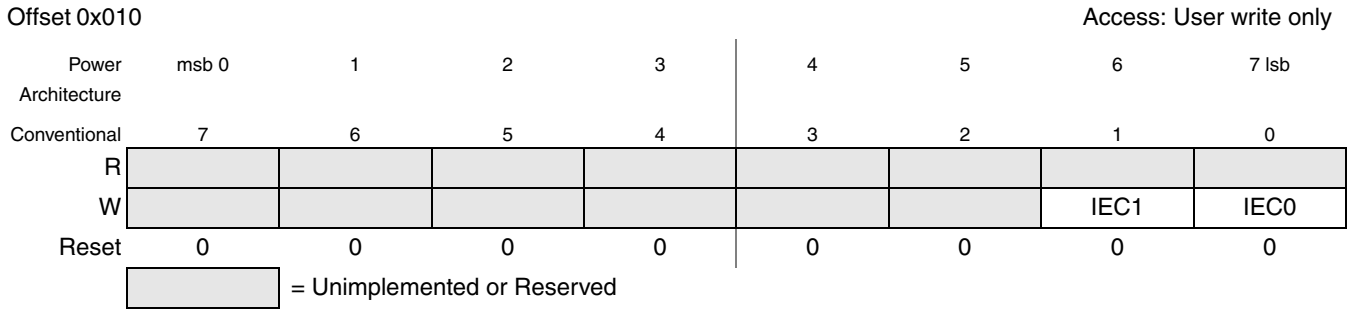


Figure 30-19. Auxiliary Control Register for all Modes (ACR)

Table 30-12. ACR Field Descriptions (continued)

Field	Description
IEC0	Interrupt Enable Control for D_CTS. 0 D_CTS has no effect on the IPC in the ISR. 1 When the D_CTS becomes high, IPC bit in the ISR sets (causing an interrupt if mask is set). After enabling the PSC, the D_CTS bit can be set. Therefore, it's important to clear the D_CTS bit before enabling this interrupt.

30.2.1.10 Interrupt Status Register (ISR)

The read-only **ISR** register provides status for all potential interrupt sources. Register contents are masked by the **IMR**.

- If an **ISR** flag sets and the corresponding **IMR** bit is also set, the internal interrupt output is asserted.
- If the corresponding **IMR** bit is cleared, the **ISR** bit state has no effect on the interrupt output.

Offset 0x14

Access: User read only

Power	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Architecture																		
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
R	IPC			ORERR	TxEMP	DB				Error	Time Out							
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-20. Interrupt Status Register for UART Mode

Offset 0x14

Access: User read only

Power	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Architecture																		
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
R	IPC			ORERR	URERR					Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT		
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 30-21. Interrupt Status Register for Other Modes
(Register is repeated for reference.)**

Table 30-13. ISR Field Descriptions (Sheet 1 of 3)

Field	Description
IPC	Input port change interrupt. 0 No IPC event has occurred. 1 An IPC event has occurred.
ORERR	Overrun Error This bit is identical to the ORERR bit in the SR register. To clear this interrupt use the reset error status command in the CR register.
TxEMP/ URERR	UART. TxEMP This bit is identical to the URERR bit in the SR register. Other Modes. Underrun Error This bit is identical to the URERR bit in the SR register. To clear this interrupt use the reset error status command in the CR register.
DB	UART. Delta Break Receiver detected an Delta Break state. Other Modes. Reserved

Programmable Serial Controller (PSC)

Offset 0x14

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	IPC				ORERR	TxEMP	DB				Error	Time Out						
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 30-20. Interrupt Status Register for UART Mode

Offset 0x14

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	IPC			ORERR	URERR						Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT	
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 30-21. Interrupt Status Register for Other Modes
(Register is repeated for reference.)**

Table 30-13. ISR Field Descriptions (Sheet 2 of 3)

Field	Description
Reserved	Other Modes. Reserved.
Error	Error This bit is identical to the error bit in the SR register. To clear this interrupt, use the reset error status command in the CR register.
Time Out	UART. Time Out This bit is identical to the TimeOut bit in the SR register. Other Modes. Reserved
CMD_SEND	AC97 Mode. Command Send ready This bit is identical to the CMD_SEND bit in the SR register. To clear this interrupt use the reset error status command in the CR register. other Modes. Reserved.
DATA_OVR	AC97 Mode. Receive Data Overwrite This bit is identical to the DATA_OVR bit in the SR register. To clear this interrupt, use the reset error status command in the CR register. Other Modes. Reserved.
DATA_VALID	AC97 Mode. Received Status Data This bit is identical to the DATA_VALID bit in the SR register. To clear this interrupt, use the reset error status command in the CR register. Other Modes. Reserved.

Offset 0x14

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	IPC				ORERR	TxE	MP	DB			Error	Time Out						
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 30-20. Interrupt Status Register for UART Mode

Offset 0x14

Access: User read only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	IPC			ORERR	URERR						Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT	
W																		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

 Figure 30-21. Interrupt Status Register for Other Modes
(Register is repeated for reference.)

Table 30-13. ISR Field Descriptions (Sheet 3 of 3)

Field	Description
UNEX_RX_SLOT	AC97 Mode. Unexpected RX Slots detect This bit is identical to the UNEX_RX_SLOT bit in the SR register. To clear this interrupt, use the reset error status command in the CR register. Other Modes. Reserved.

30.2.1.11 Interrupt Mask Register (IMR)

The write-only **IMR** register selects corresponding bits in the **ISR** that cause an interrupt.

- If one **ISR** bit is set and the corresponding IMR bit is also set, the internal interrupt output is asserted.
- If the corresponding bit in IMR is 0, the state of the **ISR** bit has no effect on the interrupt output. The **IMR** does not mask reading the **ISR**.

Offset 0x14 Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	TxEMP	DB				Error	Time Out						
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 30-22. Interrupt Mask Register for UART Mode (IMR)

Offset 0x14 Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	URERR					Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 30-23. Interrupt Mask Register for Other Modes (IMR)
(Registers are repeated for reference.)

Table 30-14. IMR Field Descriptions (Sheet 1 of 3)

Field	Description
IPC	Input port change interrupt. 0 IPC has no effect on the interrupt. 1 Enable the interrupt for IPC in the ISR register.
ORERR	Overrun Error 0 ORERR has no effect on the interrupt. 1 Enable the interrupt for ORERR
TxEMP/ URERR	UART. TxEMP 0 TxEMP has no effect on the interrupt. 1 Enable the interrupt for TxEMP Other Modes. Underrun Error. 0 URERR has no effect on the interrupt. 1 Enable the interrupt for URERR.

Offset 0x14

Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	TxEMP	DB				Error	Time Out						
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-22. Interrupt Mask Register for UART Mode (IMR)

Offset 0x14

Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	URERR					Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

 Figure 30-23. Interrupt Mask Register for Other Modes (IMR)
 (Registers are repeated for reference.)

Table 30-14. IMR Field Descriptions (Sheet 2 of 3)

Field	Description
DB	UART. Delta Break 0 DB has no effect on the interrupt. 1 Enable the interrupt for DB Other Modes. Reserved
Reserved	Other Modes. Reserved.
Error	Error 0 Error bit in the ISR register has no effect on the interrupt. 1 Enable the interrupt for Error.
Time Out	UART. Time Out 0 Time Out has no effect on the interrupt. 1 Enable the interrupt for Time Out Other Modes. Reserved
CMD_SEND	AC97 Mode. Command Send ready 0 CMD_SEND bit in the ISR register has no effect on the interrupt. 1 Enable the interrupt for CMD_SEND Other Modes. Reserved.
DATA_OVR	AC97 Mode. Receive Data Overwrite 0 DATA_OVR bit in the ISR register has no effect on the interrupt. 1 Enable the interrupt for DATA_OVR Other Modes. Reserved.

Programmable Serial Controller (PSC)

Offset 0x14

Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	TxEMP	DB				Error	Time Out						
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-22. Interrupt Mask Register for UART Mode (IMR)

Offset 0x14

Access: User write only

Power Architecture	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	lsb
Conventional		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																		
W		IPC			ORERR	URERR					Error			CMD_SEND	DATA_OVR	DATA_VALID	UNEX_RX_SLOT	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-23. Interrupt Mask Register for Other Modes (IMR)
(Registers are repeated for reference.)

Table 30-14. IMR Field Descriptions (Sheet 3 of 3)

Field	Description
DATA_VALID	AC97 Mode. Received Status Data 0 DATA_VALID bit in the ISR register has no effect on the interrupt. 1 Enable the interrupt for DATA_VALID Other Modes. Reserved.
UNEX_RX_SLOT	AC97 Mode. Unexpected RX Slots detect 0 UNEX_RX_SLOT bit in the ISR register has no effect on the interrupt. 1 Enable the interrupt for UNEX_RX_SLOT Other Modes. Reserved.

30.2.1.12 Counter Timer Upper Register (CTUR)

This write-only register holds the upper bytes of the preload value used by the timer to provide a given baudrate. Reading from this register shows the current value of the baudrate generation counter. For a detailed description, see [Section 30.2.1.13, “Counter Timer Lower Register \(CTLR\)”](#).

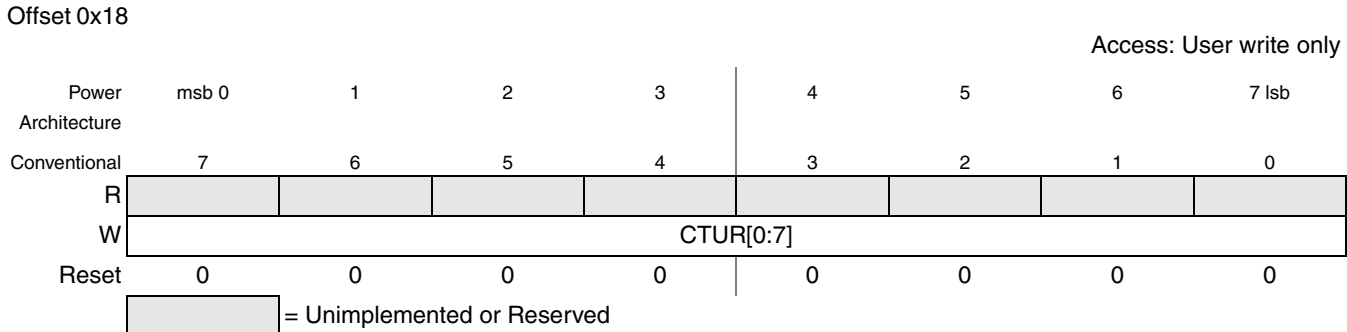


Figure 30-24. Counter Timer Upper Register for all Modes

Table 30-15. CTUR Field Descriptions

Field	Description
CTUR	Code. Frame Sync width, define the number of bit clocks during the FrameSync signal is active. FrameSync Width equals CTUR[0:7]+1 UART/ SPI. Baudrate prescaler value. See next section, Section 30.2.1.13, “Counter Timer Lower Register (CTLR)” . Other. Reserved.

30.2.1.13 Counter Timer Lower Register (CTLR)

This write-only register holds the lower bytes of the preload value used by the timer to provide a given baudrate. Reading from this register shows the current value of the baudrate generation counter.

Offset 0x1C

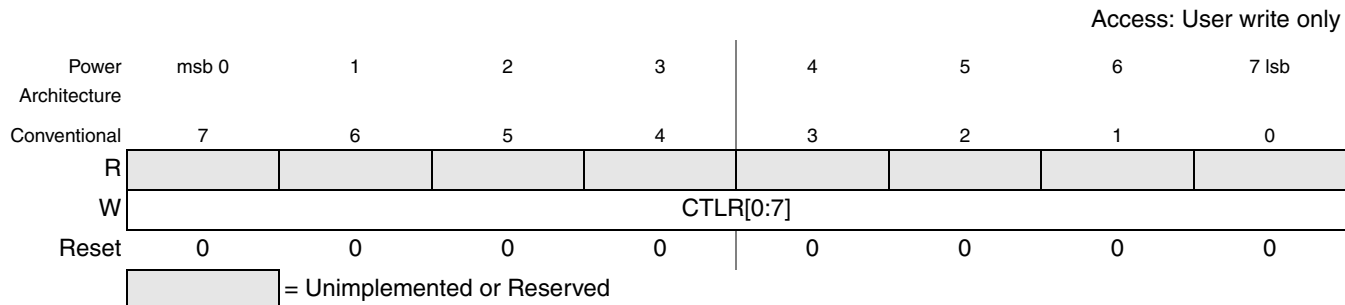


Figure 30-25. Counter Timer Upper Register for all Modes (CTLR)

Table 30-16. CTLR Field Descriptions

Field	Description
CTLR	<p>UART. Baudrate prescale value. The baudrate is calculated as:</p> <div>$\text{Baudrate} = \frac{\text{IPS_CLK frequency}}{\text{CT}[0:15] \times \text{prescaler}}$<p>where: CT[0:7] = CTUR[0:7] CT[8:15] = CTLR[0:7]</p></div> <p>The minimum CT value is 1; 0 denotes counter stop. The prescaler was defined in the CSR register.</p> <p>SPI. Delay After Transfer (DTL)</p> <div>$\text{DTL} = \frac{\text{CT}[0:15] + 2}{\text{IPS_CLK frequency}} + \frac{3}{\text{MCLK frequency}}$<p>where: CT[0:7] = CTUR[0:7] CT[8:15] = CTLR[0:7]</p></div> <p>Other. Reserved.</p>

30.2.1.14 Codec Clock Register (CCR)

This register defines the divider for the FrameSync and BCLK generation for codec mode. This register value only has effect if the GenClk bit in the PSC control register [SICR](#) was set to one or the UART mode was selected.

Offset 0x20

Access: User read/write

Power	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																	
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R		FrameSyncDiv[0:7]								BCLKDiv[8:15]							
W																	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Power		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Architecture																	
Conventional		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R		BCLKDiv[0:7]															
W																	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-26. Codec Clock Register for Codec Mode (CCR)

Offset 0x20

Access: User read/write

Power	msb	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																	
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R										BCLKDiv[8:15]							
W																	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Power		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Architecture																	
Conventional		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R		BCLKDiv[0:7]															
W																	
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-27. Codec Clock Register for UART Modes (CCR)

Table 30-17. CCR Field Descriptions

Field	Description
FrameSyncDiv	<p>Codec. Frame Sync Divider</p> <p>FrameSync is generated internally by dividing the bit clock. The FrameSyncDiv defines the number of bit clock cycles between two active frame edges:</p> $\text{FrameSync Length} = \text{FrameSyncDiv}[0:7] + 1$ <p>For more information, see Section 30.5.2.3, “Transmitting and Receiving in Soft Modem Codec Mode”</p> <p>Codec/SPI. Delay before SCK (DSCKL)</p> <p>When the PSC is in SPI mode (SICR[SPI] = 1), the FrameSyncDiv divider is used to determine the length of time the PSC delays after SS goes low/active before the first SCK transition of the serial transfer. This is a feature that exists in a QSPI. The following equation determines the actual delay before SCK:</p> $\text{DSCKL delay} = \frac{\text{FrameSyncDiv}[0:7] + 1}{\text{MCLK Frequency}}$ <p>Other Modes. Reserved</p> <p>The value 0x00 stops this counter and disables the clock generator.</p>
BCLKDiv	<p>Codec. Bit Clock Divider</p> <p>Bit clock is generated internally by dividing the MCLK frequency as follows:</p> $\text{BCLK frequency} = \frac{\text{MCLK Frequency}}{\text{BCLKDiv}[0:15] + 1}$ <p>Codec SPI. Baudrate</p> <p>SCK is generated internally by dividing the MCLK frequency as follows:</p> $\text{SCK frequency} = \frac{\text{MCLK Frequency}}{\text{BCLKDiv}[0:15] + 1}$ <p>The minimum BCLKDiv for SPI mode is 3.</p> <p>UART Modes. Time Out count value</p> <p>BCLKDiv[0:15] defines the number of UART clock events before the Time_Out event occurred if enabled.</p> <p>other Modes. Reserved</p>

30.2.1.15 AC97 Slots Register (AC97Slots)

This write only register defines which slots are expected in a receive AC97 frame and which slots are sent in a AC97 transmit frame. If the received frame doesn't match the expected slots, the [SR\[UNEXP_RX_SLOTS\]](#) bit is set. This register has only affects if the AC97 mode is selected in the [SICR](#) register and if the EnAC97 bit is active.

Offset 0x24

Access: User write only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-28. AC97 Slots Register (AC97Slots)

Table 30-18. AC97Slots Field Descriptions

Field	Description
TX_Slots	AC97 Mode. Expected Transmit Slots The bits in this register specify which data slots [3:12] are sent in an AC97 transmit frame. Each bit represents one data slot. The AC97 transmitter uses this information to generate the slot0 and reads out the according number of data words from the TXFIFO. If the TXFIFO is empty, an empty AC97 frame is sent until new data is available. Other Modes. Reserved.
RX_Slots	AC97 Mode. Expected Receive Slots The bits in this register specifies which data slots [3:12] in the receive AC97 frame must contain valid data. The AC97 codec selects the valid data slots by setting the according data valid bit in slot0[12:3]. Each bit represents one data slot. If the received valid slots do not match the expected slots, the unexpected slot received state occurs. See register SR. The received data is written to the RXFIFO only if the received slots matched the expected slots. Other Modes. Reserved.

30.2.1.16 AC97 Command Register (AC97CMD)

This register contains the AC97 address for transmit slot1 and the AC97 command data for transmit slot 2. A write access to any byte of this register sets the [SR\[CMD_SEND\]](#) bit to one. The AC97 transmitter generates a frame with valid slot1 and slot2 and pastes the values of this register to the next transmitted slot1 and slot2. After the data was sent, the [SR\[CMD_SEND\]](#) bit is cleared by the transmitter.

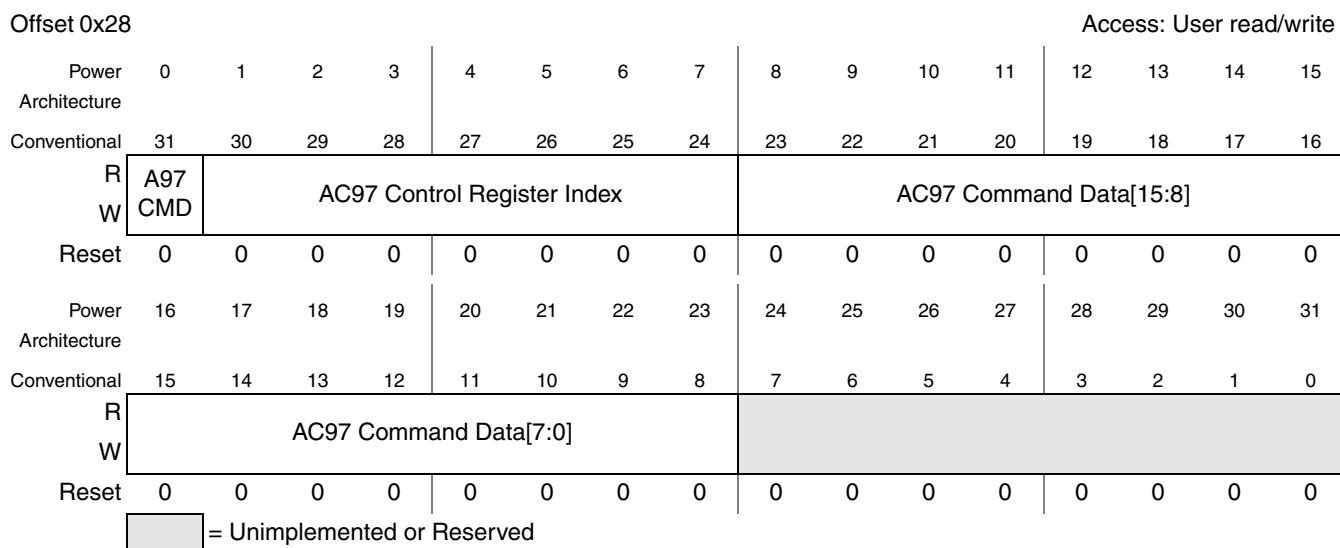


Figure 30-29. AC97 Command Register

Table 30-19. AC97CMD Field Descriptions

Field	Description
AC97 CMD	AC97 Mode. AC97 Command This bit indicates if the access to the control register is a read or write access. It is pasted to the slot1 bit 19. 0 Write access 1 Read access Other Modes. Reserved.
AC97 Control Register Index	AC97 Mode. AC97 Address Register This register contains target control register address. It is pasted to the slot1 bit 18 to 12. Other Modes. Reserved.
AC97 Command Data	AC97 Mode. AC97 Command Data Register This register defines the command data value for a write command. It is pasted to the slot2 bit 19 to 4. Other Modes. Reserved.

30.2.1.17 AC97 Status Data Register (AC97Data)

This read-only register contains the received response of an AC97 read command. If this register contains new data, the [SR\[DATA_VALID\]](#) is set to one by the receiver. A read access to this register clears the [SR\[DATA_VALID\]](#) bit.

Offset 0x2C

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AC97 Register Index Echo								AC97 Control Register Read Data[15:8]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AC97 Control Register Read Data[7:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 30-30. AC97 Status Register (AC97Data)

Table 30-20. AC97Data Field Descriptions

Field	Description
AC97 Register Index Echo	AC97 Mode. AC97 Register Index Echo This register contains the received register index echo from the Rx slot 0. Other Modes. Reserved.
AC97 Control Register ReadData	AC97 Mode. AC97 Control Register Read Data This register contains the received control data from Rx slot 2. Other Modes. Reserved.

30.2.1.18 Input Port Register (IP)

This read-only IP register shows the current state of the input ports.

Offset 0x034

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7 lsb
Conventional	7	6	5	4	3	2	1	0
R							DCD	CTS
W								
Reset	1	1	1	1	1	1	0	0
	= Unimplemented or Reserved							

Figure 30-31. Input Port Register for UART Modes (IP)

Offset 0x034

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7 lsb
Conventional	7	6	5	4	3	2	1	0
R		TGL					DCD	CTS
W								
Reset	1	0	1	1	1	1	0	0
	= Unimplemented or Reserved							

Figure 30-32. Input Port Register for Codec Mode (IP)

Offset 0x34

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7 lsb
Conventional	7	6	5	4	3	2	1	0
R	LPWR	TGL					DCD	CTS
W								
Reset	1	1	1	1	1	1	0	0
	= Unimplemented or Reserved							

Figure 30-33. Input Port Register for AC97 Mode (IP)

Table 30-21. IP Field Descriptions

Field	Description
LPWR	AC97. Low-power mode in AC97 mode 0 Codec is in low power mode. 1 Normal operation. Other Modes. Reserved
TGL	AC97/Codec. Test usage. Toggle by FrameSync. Other Modes. Reserved.
DCD	Current state of the $\overline{\text{DCD}}$ input. 0 $\overline{\text{DCD}}$ input is low. 1 $\overline{\text{DCD}}$ input is high.

Offset 0x034

Access: User read only

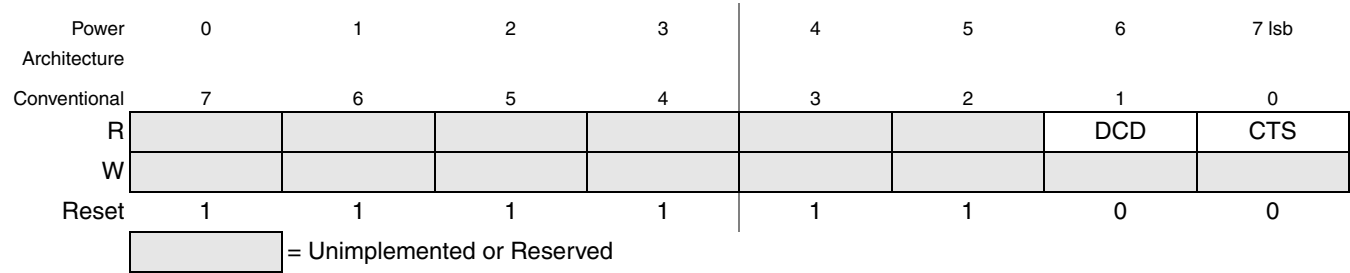


Figure 30-31. Input Port Register for UART Modes (IP)

Offset 0x034

Access: User read only

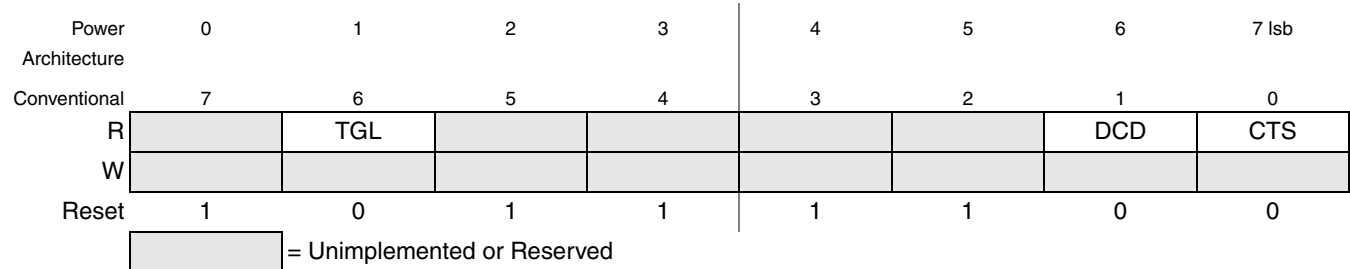


Figure 30-32. Input Port Register for Codec Mode (IP)

Offset 0x34

Access: User read only

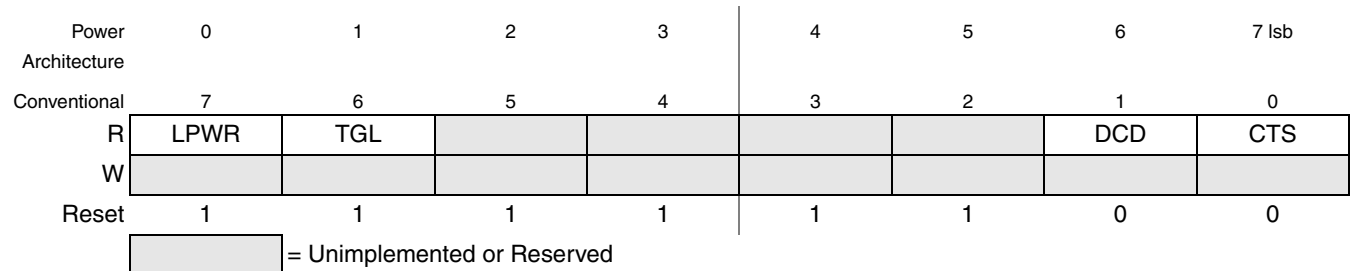


Figure 30-33. Input Port Register for AC97 Mode (IP)

Table 30-21. IP Field Descriptions (continued)

Field	Description
CTS	Current state of the $\overline{\text{CTS}}$ input 0 Input port $\overline{\text{CTS}}$ is low. 1 Input port $\overline{\text{CTS}}$ is high.

30.2.1.19 Output Port 1 Bit Set (OP1)

This is a write-only register. Output ports are asserted by writing to this register.

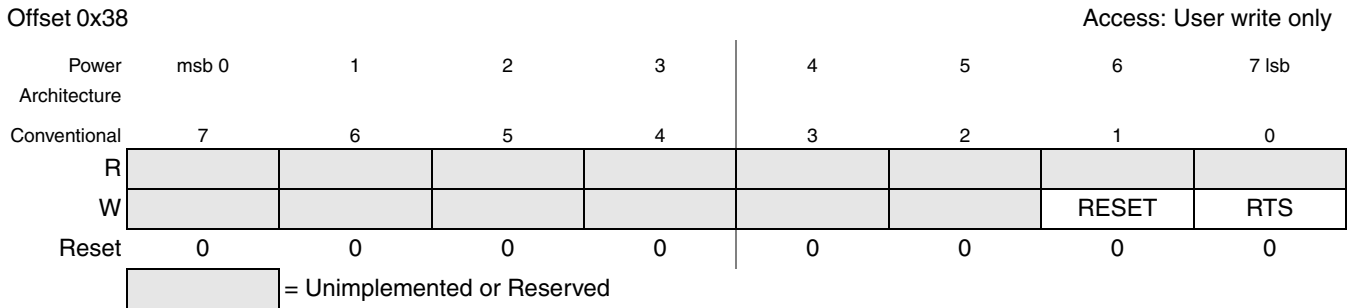


Figure 30-34. Output Port 1 Bit Set Register for all Modes (OP1)

Table 30-22. OP1 Field Descriptions

Field	Description
RESET	AC97. Assert RESET output. 0 No operation 1 Asserts output port RESET, (low active signal RESET becomes 0). Other Modes. Reserved.
RTS	AC97. Reserved Other Modes. Assert RTS output. 0 No operation 1 Asserts output port $\overline{\text{RTS}}$, (low active signal $\overline{\text{RTS}}$ becomes 0).

30.2.1.20 Output Port 0 Bit Set (OP0)

This is a write-only register. Output ports are negated by writing to this register.

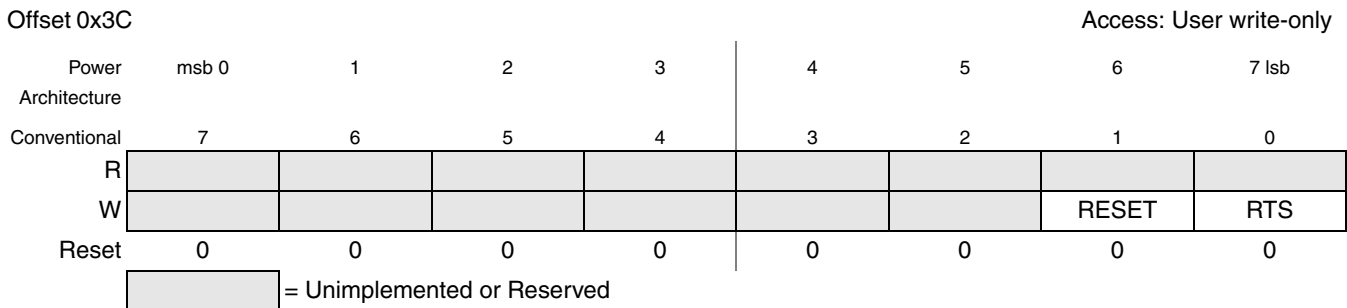


Figure 30-35. Output Port 0 Bit Set Register for all Modes

Table 30-23. OP0 Field Descriptions

Field	Description
RESET	AC97. Assert RESET output. 0 No operation 1 Negates output port RESET, (low active signal RESET becomes 1). Other Modes. Reserved.
RTS	AC97. Reserved Other Modes. Assert $\overline{\text{RTS}}$ output. 0 No operation 1 Negates output port $\overline{\text{RTS}}$, (low active signal $\overline{\text{RTS}}$ becomes 1).

30.2.1.21 Serial Interface Control Register (SICR)

This register sets the main operation mode.

Offset 0x40					Access: User read/write			
Power Architecture	msb 0	1	2	3	4	5	6	7
Conventional	7	6	5	4	3	2	1	0
R	ACRB	AWR	DTS1	SHDIR	SIM[3:0]			
W								
Reset	0	0	0	0	0	0	0	0
Power Architecture	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8
R	GenClk	I2S	ClkPol	SyncPol			ESAI	EnAC97
W								
Reset	0	0	0	1	0	0	0	1
Power Architecture	16	17	18	19	20	21	22	23 lsb
Conventional	23	22	21	20	19	18	17	16
R	SPI	MSTR	CPOL	CPHA	UseEOF			En_OutBuf
W								
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Figure 30-36. Serial Interface Control Register for all Modes (SICR)

Table 30-24. SICR Field Descriptions

Field	Description
ACRB	<p>AC97. AC97 Cold Reset to the transceiver in PSC. This bit was prepared for backward compatibility with the MCF5407 USART. It is recommended to use OP1 and OP0 registers to set and to reset AC97 reset line.</p> <p>0 The transceiver recovers from low power mode in AC97. 1 The transceiver stays in the current state.</p> <p>Other Modes. Reserved.</p>
AWR	<p>AC97. AC97 Warm Reset (to the PSC and off-chip AC97 Codec)</p> <p>0 AC97 warm reset is negated. Output functions normally as the AC97 Sync. 1 Force 1 on Sync output to recover the AC97 interface from AC97 power down mode.</p> <p>Other Modes. Reserved.</p>
DTS1	<p>Codec. Delay of time slot #1.</p> <p>0 First bit of first time slot of a new frame starts at the rising edge of FrameSync. 1 First bit of first time slot of a new frame starts one bit clock cycle after the rising edge of FrameSync.</p> <p>Other Modes. Reserved.</p>

Table 30-24. SICR Field Descriptions (continued)

Field	Description
SHDIR	Codec. Shift Direction. 0 msb first 1 lsb first Other Modes. Reserved.
SIM[3:0]	PSC operation mode. CAUTION: When the operating mode change occurs, all Rx/Tx and error statuses are reset. Rx and Tx are disabled. 0000 = UART mode, \overline{DCD} input ignored 1000 = UART mode, \overline{DCD} input is effective 0001 = Codec mode, 8-bit data 1001 = Codec mode, 12-bit data 0010 = Codec mode, 16-bit data 1010 = Codec mode, 20-bit data x011 = AC97 mode 0100 = Reserved 1100 = Reserved x101 = Reserved x110 = Reserved 0111 = Codec mode, 24-bit data 1111 = Codec mode, 32-bit data
GenClk	Codec. Generate Bit Clock and FrameSync. Not used to enable the SPI master mode. This bit must also be set to the MSTR to enable the SPI master mode 0 Use bit clock and FrameSync provided by external device 1 Use bit clock and FrameSync generated internally from MCLK Other Modes. Reserved
I2S	Codec. I2S mode 0 No I2S mode supported 1 PSC works in I2S mode Other Modes. Reserved.
ClkPol	Codec. Bit Clock Polarity 0 Data in is sampled on the falling edge of the BCLK and data out is shifted on the rising edge 1 Data in is sampled on the rising edge of the BCLK and data out is shifted on the falling edge Other Modes. Reserved.
SyncPol	Codec. FrameSync Polarity, must be cleared during SPI mode 0 FrameSync is low true 1 FrameSync is high true Codec I2S. FrameSync Polarity 0 Frame starts if LRCK is low 1 Frame starts if LRCK is high Other Modes. Reserved.
ESAI	Codec. Enhanced Serial Audio Interface 0 PSC doesn't support the ESAI mode. 1 PSC supports the ESAI mode. This mode allows the PSC to send and receive more the one data word per frame, if the frame length is greater than the word length. The PSC send only complete data words. Other Modes. Reserved.

Table 30-24. SICR Field Descriptions (continued)

Field	Description
EnAC97	<p>Codec. Normal AC97 mode. Takes affect only when the AC97 mode is selected (SIM = 0x3)</p> <p>0 If AC97 mode was selected, Legacy AC97 mode used</p> <p>1 If AC97 mode was selected, AC97 mode transmits and receives the data.</p> <p>Other Modes. Reserved.</p>
SPI	<p>Codec. SPI mode</p> <p>0 PSC does not behave like an SPI</p> <p>1 PSC behaves like an SPI</p> <p>Other Modes. Reserved.</p>
MSTR	<p>Codec. SPI Master mode. Takes affect only when bit SICR[SPI mode] equals 1. Also, the GenClk bit must be set to enable the clock generation behavior of the SPI master mode</p> <p>0 PSC behaves as an SPI slave</p> <p>1 PSC behaves as an SPI master</p> <p>Other Modes. Reserved.</p>
CPOL	<p>Codec. SPI Clock Polarity. takes effect only when bit SICR[SPI mode] equal 1</p> <p>This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values</p> <p>0 Active-low clocks selected; SCK idles high</p> <p>1 Active-high clocks selected; SCK idles low</p> <p>Other Modes. Reserved.</p>
CPHA	<p>Codec. SPI Clock Phase</p> <p>This bit is used to shift the SCK serial clock. To transmit data between SPI modules, the SPI modules must have identical CPHA values</p> <p>0 Data transfer starts which assertion of \overline{SS}</p> <p>1 Data transfer starts with the first edge of SCK</p> <p>Other modes. Reserved.</p>
UseEOF	<p>Codec. Use End-of-Frame flag takes effect only when SPI mode is selected</p> <p>0 One, two, or four bytes are transferred while slave select (SS) is held low, as determined by SICR[SIM]</p> <p>1 Multiple bytes are transferred while maintaining SS low, up to and including the next byte read from the Tx FIFO that has its EOF flag set</p> <p>Other modes. Reserved.</p>
En_OutBuf	<p>Enable Output Buffer</p> <p>0 The output logic is only enabled if the receiver or transmitter is enabled. After enabling the transmitter or receiver, the internal generated signals are visible on the output of the device.</p> <p>1 The output logic is enabled by setting is bit enabled. After setting this bit, the internal generated signals are visible on the output of the device.</p>

30.3 PSC Functions Overview

The PSC module provides different groups of interfaces to connect to different serial devices.

Figure 30-37. shows the groups of interfaces.

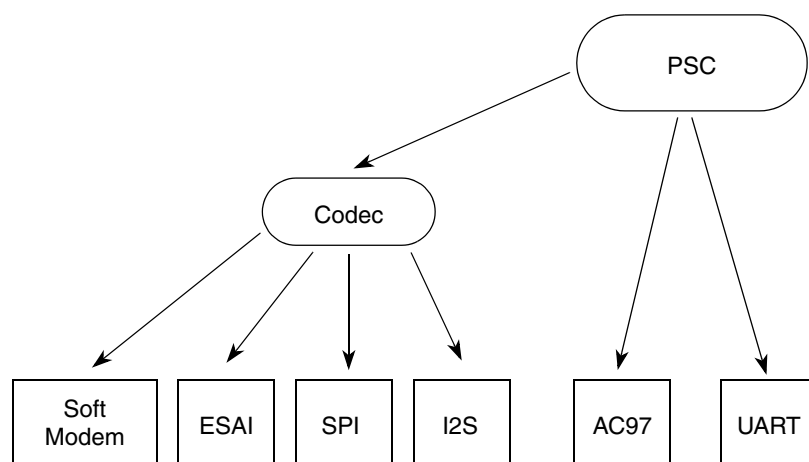


Figure 30-37. PSC Functions Overview

1. **PSC Codec Mode:** In this section, the name codec mode is used as a collective term for the normal soft modem, the SPI, I2S, and ESAI mode. This interface is provided by one internal block of the PSC. The interface consists of serial Tx and Rx lines, a bit clock line and a FrameSync signal. For these modes, the clock configuration is similar and uses the same configuration registers. The transmitter converts the parallel data from the CPU to a serial bit-stream, and the receiver converts the serial data from the Rx line to parallel data. The PSCs support codec mode with 8, 12, 16, 20, 24, and 32 bit data width, with active high or active low FrameSync signal and programmable bit clock polarity. All codec modes can work as an codec master (PSC drive the bit clock and FrameSync signals) or as a codec slave (PSC receives the bit clock and frame sync signal). For more information about the codec mode, see [Section 30.5.2, “PSC in Codec Mode”](#).
2. **AC97 Mode:** When programmed as AC97, the PSC works as an AC97 controller. This means the PSC receives the BCLK from the external AC97 codec and provides the FrameSync signal to the external codec. If the PSC detects a codec not ready status, the PSC stops sending and receiving data. In AC97 mode, only the used data slots must be in the FIFO. The PSC generates the slot0, slot1, and slot2 values depending on data to send. In AC97 modes, the PSC reads only 32 bits from the FIFO. For more information about the AC97 mode, see [Section 30.5.3, “PSC in AC97 Mode”](#).
3. **PSC UART Mode:** When programmed as a UART, the PSC serial communication channel provides a full-duplex asynchronous receiver and transmitter deriving the operating frequency from an internal clock. The transmitter converts parallel data from the CPU to a serial bit-stream, inserting appropriate start, stop, and parity bits. It outputs the resulting stream on the channel transmitter serial data output (TxD). The receiver converts serial data from the channel receiver serial data input (RxD) to parallel format, checks for start, stop, and parity bits, or line break conditions, and transfers the assembled character onto the bus during read operations. The receiver may be poll-driven or interrupt-driven. For more information about the UART mode, see [Section 30.5.1, “PSC in UART Mode”](#).

Table 30-25. PSC Pin Assignment Versus Operating Mode

	UART	AC97	SPI		Codec TCM/ESAI					I ² S				
					0 Tx, 3 Rx	1 Tx, 2 Rx	1 Tx, 1 Rx	2 Tx, 1 Rx	3 Tx, 0 Rx	0 Tx, 3 Rx	1 Tx, 2 Rx	1 Tx, 1 Rx	2 Tx, 1 Rx	3 Tx, 0 Rx
PSCX_0	RTS	BCLK	SCLK		BCLK					SCLK				
PSCX_1	CTS	SYNC	SS		FRAMESYNC					LRCK				
PSCX_2	TxD	SDATA_ OUT	MOSI	MISO	RxD3	TxD				RxD3	TxD			
PSCX_3	RxD	SDATA_ IN	MISO	MOSI	RxD				TxD3	RxD				TxD3
PSCX_4	DCD	RESET	MCLK		RxD2		MCLK	TxD2		RxD2		MCLK	TxD2	

NOTE

For Codec TCM/ESAI and I2S, the BCLK and SCLK pins are outputs when the port is configured as a master and inputs when the port is configured as a slave.

For Codec TCM/ESAI and I2S, the SYNC and LRCK pins are outputs when the port is configured as a master and inputs when the port is configured as a slave.

30.4 Features

General features:

- Each channel is programmable to normal (full-duplex), automatic echo, local loop-back, or remote loop-back mode
- Automatic wake-up mode for multidrop applications
- Six maskable interrupt conditions

PSC UART mode:

- Each is clocked by an internal clock source (IPS_CLK), eliminating the need for an external crystal
- Full-duplex asynchronous receiver/transmitter channel
- Programmable data format:
 - Five to eight data bits plus parity
 - Odd, even, no parity, or force parity
 - One, one-and-a-half, or two STOP bits
- Parity, framing, and overrun error detection
- False-start bit detection
- Line-break detection and generation
- Detection of breaks originating in the middle of a character
- Start/end break interrupt/status

PSC codec mode:

- Programmable to interface to an 8, 12, 16, 20, 24, or 32-bit Codec for soft modem support up to 3 Tx or Rx lines
- Supports master mode, driving clock, and FrameSync signals
- Supports slave mode, receiving clock, and the FrameSync from the external codec
- Supports multichannel mode, with 3 data lines in codec mode
- Supports full duplex SPI interface
- Supports I2S interface
- No parity error, framing error, or line break detection in codec mode
- Ability to generate a master clock (MCLK) for an external codec device, independent from the mode (master or slave)
- Programmable width of the FrameSync signal
- FrameSync and bit clock frequencies are independently programmable
- Frame sync and bit clock polarity are programmable

AC97 mode:

- Supports AC97 mode, only the data slots must be available, the control slot data are generated by the PSC
- Supports AC97 wake-up and power-down modes

30.5 Modes of Operation

This section describes the different PSC operation modes, including the pin muxing, the module configuration, signal definition, and some programming examples. All PSCs are independent and can be used at the same time in different modes.

30.5.1 PSC in UART Mode

Select the UART mode by writing the corresponding value to the PSC Control (SICR) register. The PSC UART mode is the default mode after reset. The important registers to configure the PSC for UART mode are:

- SICR register — select the UART mode
- CSR register — select the clock source
- CTUR, CTLR register — select the baudrate
- MR1 register — select the UART mode (parity mode, bits per character)
- MR2 register — select $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ control, Stop Bit Length
- CR register — enable or disable receiver and transmitter

30.5.1.1 Block Diagram and Signal Definition for UART Mode

The Figure 30-38 shows the simplified block diagram of the PSC for UART mode.

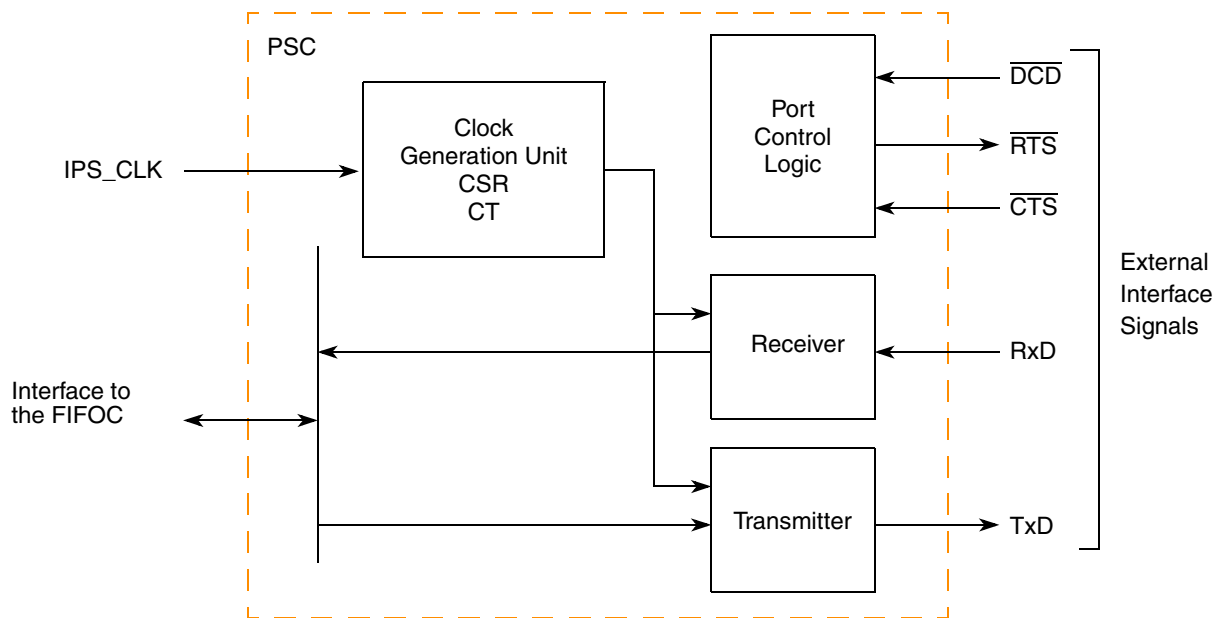


Figure 30-38. PSC UART Block Diagram

An internal interrupt request signal is provided to notify the interrupt controller of an interrupt condition. The output is the logical NOR of unmasked IMR bits. The interrupt level of a PSC module is programmed in the interrupt controller.

The PSC can automatically transfer data using the DMA, rather than interrupting the core.

Table 30-26 briefly describes the PSC module signals.

NOTE

The terms assertion and negation are used to avoid confusion between active-low and active-high signals.

- Asserted indicates a signal is active, independent of the voltage level
- Negated indicates a signal is inactive.

Table 30-26. PSC Signal Description for UART Mode

Signal	Description
TxD	Transmitter Serial Data Output. TxD is held high (mark condition) when Tx is disabled, idle, or operating in the local loop-back mode. Data is shifted out on TxD on the falling edge of the clock source, with the least significant bit (lsb) sent first.
RxD	Receiver Serial Data Input. Data received on RxD is sampled on the rising edge of the clock source, with the lsb received first.
CTS	Clear-to-Send. This input can generate an interrupt on a change of state.
RTS	Request-to-Send. This output can be programmed to be negated or asserted automatically by Rx or Tx. When connected to a transmitter CTS, RTS can control serial data flow.
DCD	Data carrier detect Input. In the enhanced UART mode, this signal must be asserted during the data transmission.

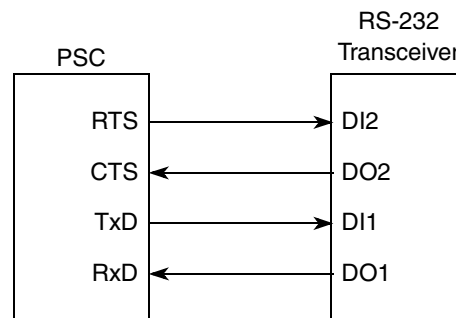


Figure 30-39. Signal Configuration for a PSC/RS-232 Interface

30.5.1.2 UART Clock Generation

IPS_CLK clock serves as the basic timing reference for the clock source generator logic, which consists of a clock generator and a programmable 16-bit divider dedicated to the PSC and a fix prescaler. The IPS_CLK clock passes through the prescaler (divide by 32 or 10) and then through the 16-bit divider of the concatenated CTUR and CTLR registers. See Figure 30-40.

Eqn. 30-1

$$\text{Baudrate} = \frac{\text{IPS_CLK}}{\text{prescaler} \times \text{CT}[0:15]}$$

Where:

$$\text{CT}[0:7] = \text{CTUR}[0:7]$$

$$\text{CT}[8:15] = \text{CTLR}[0:7]$$

For example, to generate a 9600 baudrate with 66 MHz IPS_CLK bus frequency and a predivider value of 32, the register values can be calculated as follows:

Eqn. 30-2

$$\text{Divider} = \frac{66 \text{ MHz}}{32 \times 9600} = 215(\text{decimal}) = 0x00d7$$

Therefore, **CTUR** equals 0x00 and **CTLR** equals 0xD7.

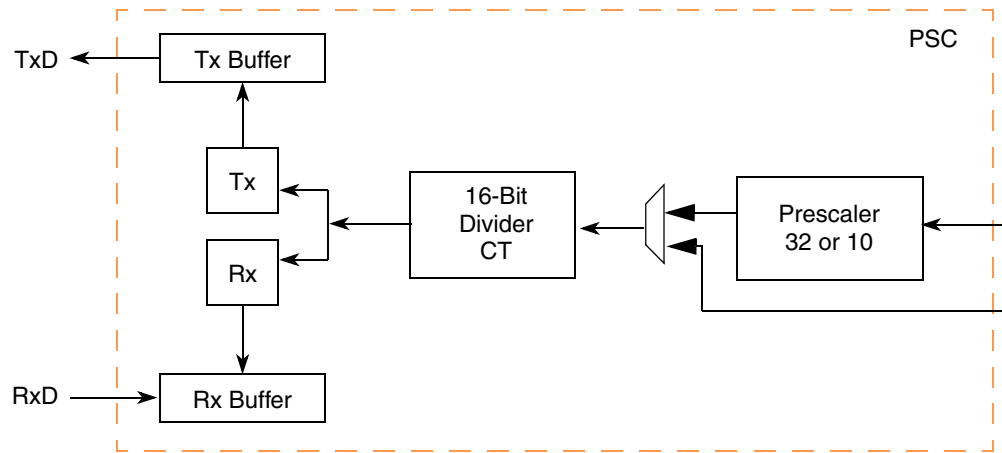


Figure 30-40. Clocking Source Diagram

30.5.1.3 Transmitting in UART Mode

After a hardware reset, all PSCs are in UART mode. The PSC command register (**CR**) enables the transmitter. The transmitter converts parallel data from the CPU to a serial bit-stream on TxD. It automatically sends a start bit followed by:

- The programmed number of data bits
- An optional parity bit
- The programmed number of stop bits

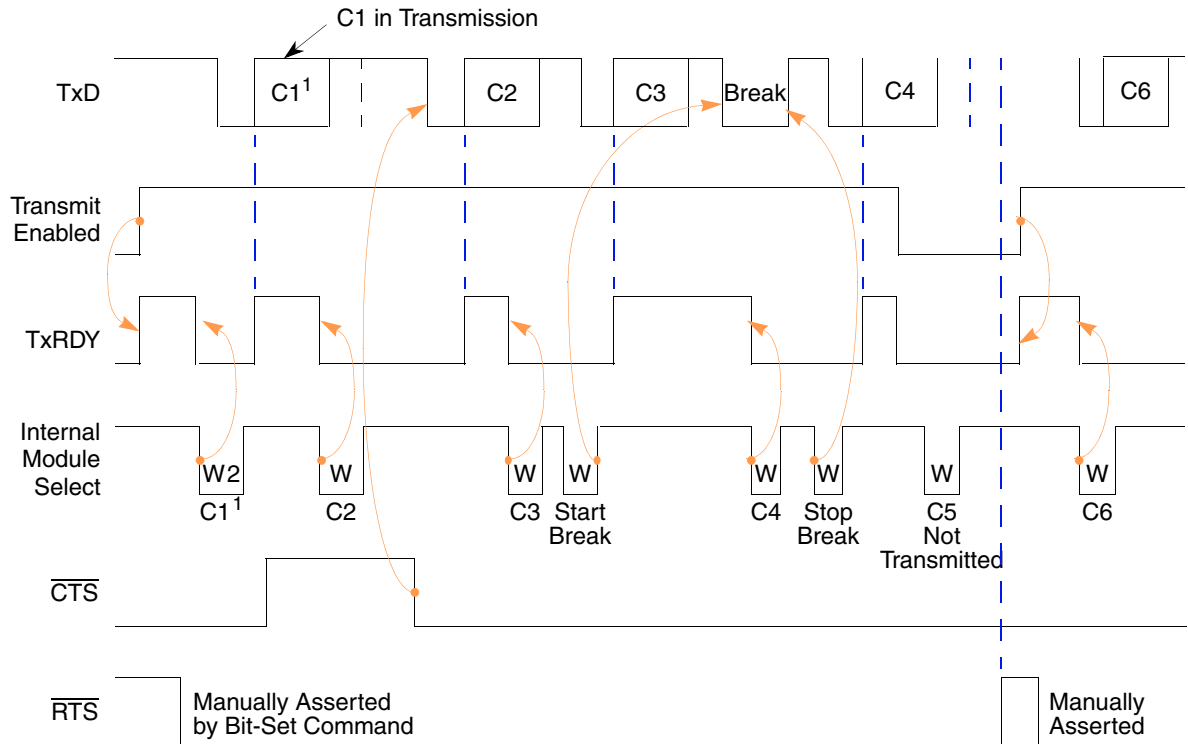
The lsb is sent first. Data is shifted from the Tx output on the falling edge of the clock source.

After the stop bits are sent, if no new character is in the Tx holding register, the TxD output remains high (mark condition) and the Tx empty bit, **SR[TxEMP]**, is set. Transmission resumes and TxEMP is cleared when the CPU loads a new character into the PSC Tx buffer (TB).

- If the transmitter receives a disable command, it continues until any character in the Tx shift register is completely sent.
- If the transmitter is reset through a software command, operation stops immediately.
- If the clear-to-send operation is enabled, $\overline{\text{CTS}}$ must be asserted for the character to be transmitted.
- If $\overline{\text{CTS}}$ is negated in the middle of a transmission, the character in the shift register is not sent and TxD remains in mark state until $\overline{\text{CTS}}$ is reasserted.
- If the transmitter is forced to send a continuous low condition by issuing a send break command, the transmitter ignores the state of $\overline{\text{CTS}}$.

- If the transmitter is programmed to automatically negate $\overline{\text{RTS}}$ when a message transmission completes, $\overline{\text{RTS}}$ must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{RTS}}$ is appropriately programmed, $\overline{\text{RTS}}$ is negated one bit-time after the character in the shift register is completely transmitted. The transmitter must be manually re-enabled by reasserting $\overline{\text{RTS}}$ before the next message is sent.

Figure 30-41 shows the transmitter functional timing information.



NOTES:

- Cn = transmit characters
- W = write
- MR2[TxCTS] = 1
- MR2[TxRTS] = 1

Figure 30-41. Timing Diagram—Transmitter

30.5.1.4 Receiving in UART Mode

After a hardware reset, all PSCs are in UART mode. The receiver is enabled through its **CR**, as described in Section 30.2.1.5, “Command Register (CR)”. Figure 30-42 shows the receiver functional timing.

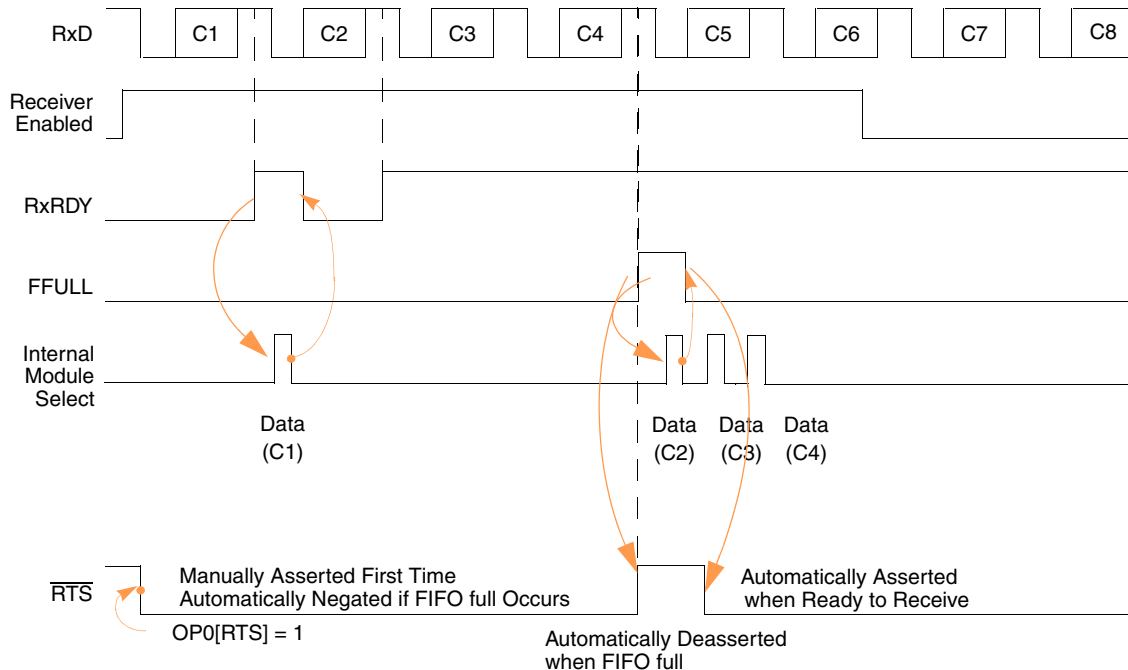


Figure 30-42. Timing Diagram – Receiver

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on RxD, the state of RxD is sampled. Starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit-time clock (synchronous operation).

- If RxD is sampled high, start bit is invalid; a valid start bit search begins again.
- If RxD remains low, a valid start bit is assumed and receiver continues sampling input at 1-bit time intervals at the theoretical center of the bit. This continues until the proper number of data bits and parity, if any, is assembled and one stop bit is detected.

RxD input data is sampled on the rising edge of the programmed clock source. The lsb is received first. Data is then transferred to a receiver holding register and RxRDY bit is set. If the character is less than eightbits, the most significant unused bits in the receiver holding register are cleared.

If the **MR1[RxRTS]** bit was set to one, control the **RTS** line by writing to the output port register. For all user generated commands to the UART receiver, like enable Rx, disable Rx, set break or read data from the RX FIFO, set the associated **RTS** signal by writing the **OP0** or **OP1** register. However, the UART receiver automatically deasserts the **RTS** signal if the RX FIFO is full.

After the stop bit is detected, the receiver immediately looks for the next start bit.

- If a non-zero character is received without a stop bit (framing error) and RxD remains low for one-half of bit period after stop bit is sampled, the receiver operates as if a new start bit were detected. Parity error (PE), framing error (FE), overrun error (ORERR), and received break (RB) conditions set respective error and break flags in [SR](#) at the received character boundary and are valid only if RxRDY is set.
- If a break condition is detected (RxD is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver shift register and [SR\[RB\]](#) is set. RxD must return to a high condition for at least one-half bit-time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character, if the break persists through the next character time.

- If the break begins in the middle of a character, the receiver places the damaged character in the Rx FIFO stack and sets the corresponding [SR](#) error bit.
- If the break lasts until the next character time, the receiver places an all zero character into the Rx FIFO and sets [SR\[RB\]](#).

30.5.1.5 TimeOut Counter Behavior

The TimeOut counter is available during UART mode only. If this behavior is enabled then an internal counter can generate an interrupt if the time after the last received data word is bigger than the programmed limit. The first received data word after enabling this behavior starts the counter. A receive data word before the counter reach the limit clears the counter value and starts the counter again. The value in the register defines the time base on number of clock events for the programmed UART clock source (32 or 10 clock events per bit depend on the selected prescaler).

30.5.1.6 Configuration Sequence for UART Mode

[Table 30-27](#) shows the configuration sequences. This list includes the UART mode related registers only, not the other configure values like interrupt and FIFO configurations. PSC module registers can be accessed by word or byte operations.

Table 30-27. General Configuration Sequence for UART Mode

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled by the work before.
CSR	0xdd00	Select the clock source
SICR	0x00000000 or 0x08000000	Select the UART mode
MR1	0xXX	Select error mode, parity mode, and the parity type
MR2	0xXX	Select channel mode, port control, and stop-bit length

Table 30-27. General Configuration Sequence for UART Mode

Register	Value	Setting
CTUR	0x00	Set the calculated baudrate depend on the IP bus clock
CTLR	0xD7	
IMR	0XXXXX	Select the desired interrupt
CR	0x05	Enable Tx and Rx

30.5.1.7 UART Multidrop Mode

Setting [MR1](#)[PM] programs the PSC to operate in a walk-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their channel receivers disabled, they continuously monitor the masters data stream. When the master sends an address character, the slave receiver channel notifies its respective CPU by setting [SR](#) and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Slave stations not addressed continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process.

[Figure 30-43](#) shows functional timing information for multidrop mode.

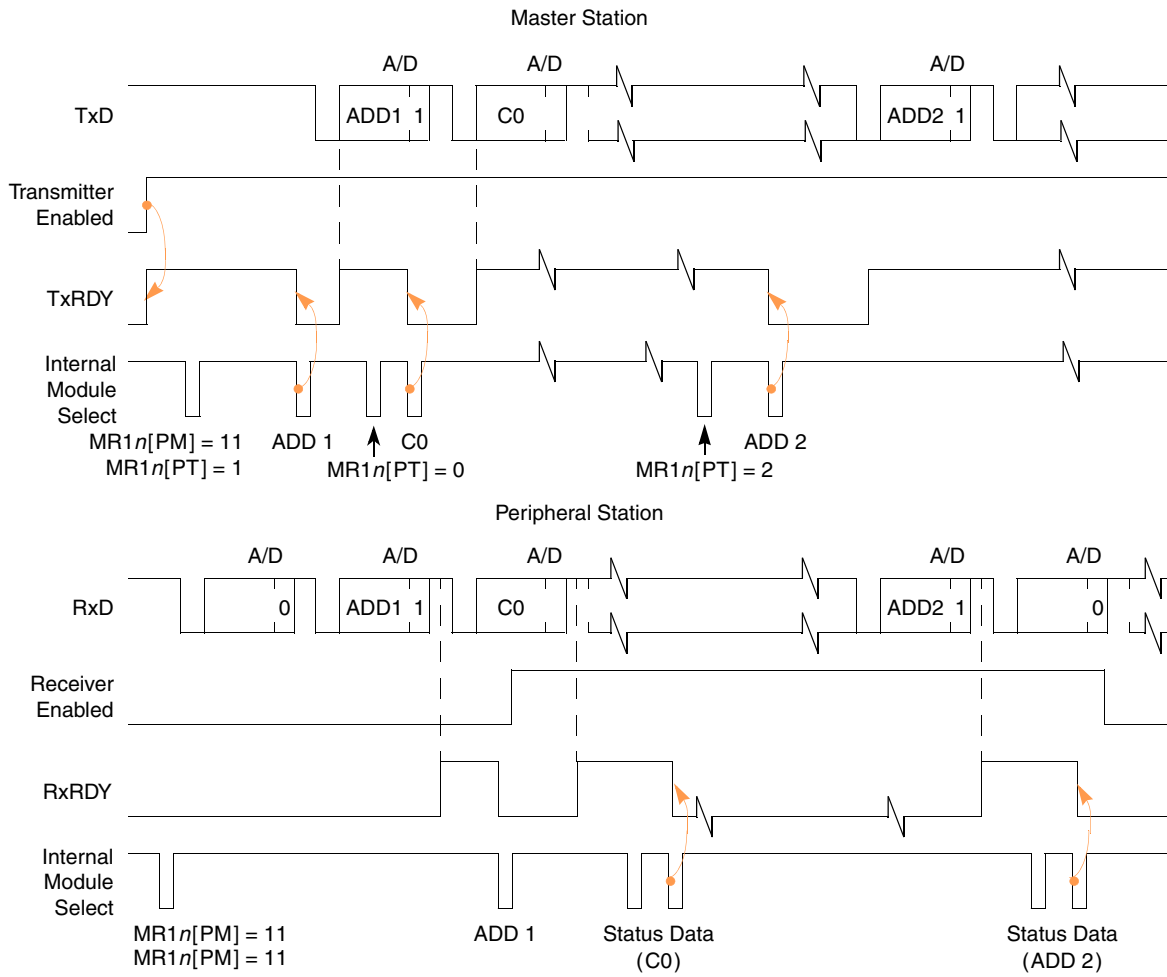


Figure 30-43. Timing Diagram – Multidrop Mode

A character sent from the master station consists of:

- A start bit
- A programmed number of data bits
- An address/data (A/D) bit flag
 - A/D=1 indicates an address character
 - A/D=0 indicates a data character
- A programmed number of stop bits

A/D polarity is selected through [MR1\[PT\]](#). [MR1](#) should be programmed before enabling the transmitter and loading the corresponding data bits into the Tx buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled.

- If the receiver is disabled, it sets the RxRDY bit and loads the character into the receive register, provided the received A/D bit is 1 (address tag). If the received A/D bit is 0 (data tag), the character is discarded.
- If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register stack during read operations.

In either case, data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error ([SR\[PE\]](#)).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit. Parity is neither calculated nor checked. Messages in this mode may continue to contain error detection and correction information. One way to provide error detection if 8-bit characters are not required is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

30.5.2 PSC in Codec Mode

After reset, all PSCs are in UART mode. The PSCs can be switched to one of the codec modes by writing the appropriate value to the [SICR](#) register. The other values should be initialized at the same time. During codec mode, the PSC can connect to codec interfaces with 8, 12, 16, 20, 24, or 32 bit data. For all these modes, the PSC can be programmed to behave as a normal soft modem interface, SPI, ESAI, or I2S interface. The PSC codec supports these modes acting as the master mode (PSC drive the BCLK and FrameSync signal) or slave mode (PSC receive the BCLK and FrameSync signals) functionality. Independently from the mode (master or slave), the PSC can provide a MCLK (master clock) for an external codec device. This behavior eliminates the need for an external crystal for the external codec device. The number of used Tx and Rx lines can be programmed via the [MR2](#) register. [Figure 30-44](#) shows a simplified block diagram for the PSC codec mode. The important register to configure the PSC for codec mode are:

- [SICR](#) register—select the codec mode
- For master mode:
 - select and enable MCLK divider
 - [CCR](#)—select BCLK and FrameSync frequency

- **CTUR**—select FrameSync width
- **CR** register—enable or disable receiver and transmitter
- **MR2** register—configure the number of Tx and Rx channels

30.5.2.1 Block Diagram and Signal Definition for Codec Mode

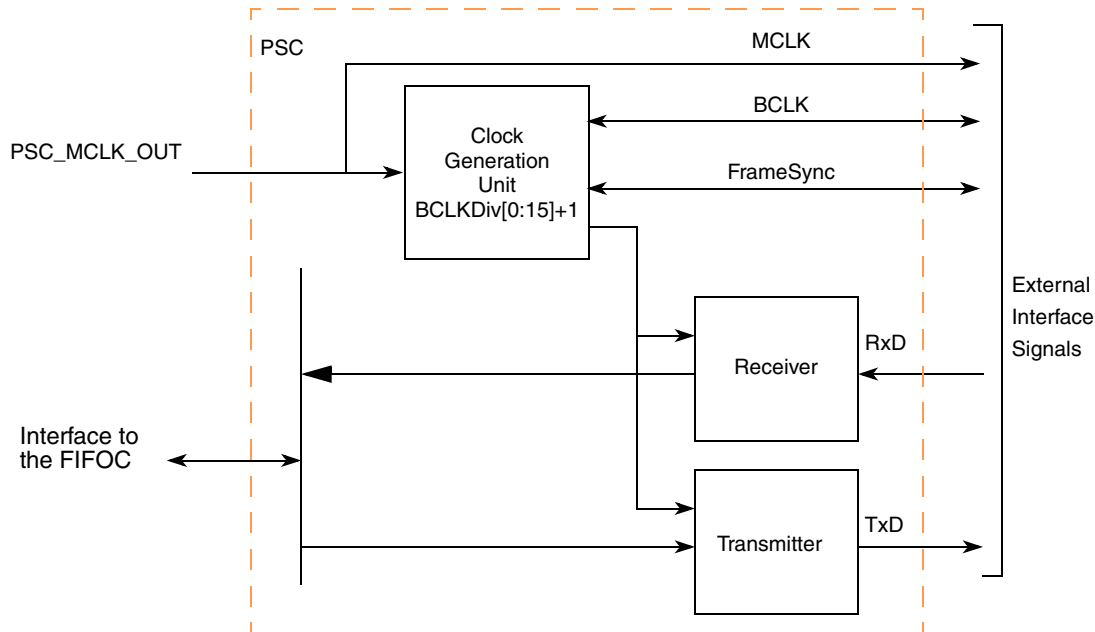


Figure 30-44. PSC Codec Block Diagram

Table 30-28. PSC Signal Description for Codec Mode

Signal	Description
TxD	Transmitter Serial Data Output. Depend on the programming of the MR2 register, up to three Tx channels are supported. Data is shifted out on TxD on the falling or rising edge of the clock source. Transfers can be specified as lsb or msb first. TxD is held low when Tx is disabled or idle. <ul style="list-style-type: none"> • Data shifted out on the rising edge of CLK if SICR[ClkPol] equals 0 • Data shifted out on the falling edge of CLK if SICR[ClkPol] equals 1 • Data send msb first if SICR[SHDIR] equals 0 • Data send lsb first if SICR[SHDIR] equals 1
RxD	Receiver Serial Data Input. Depend on the programming of the MR2 register, up to three Rx channels are supported. Data received on RxD is sampled on the falling or rising edge of the clock signal. Transfers can be specified as either lsb or msb first. <ul style="list-style-type: none"> • Data sampled on the rising edge of CLK if SICR[ClkPol] equals 1 • Data sampled on the falling edge of CLK if SICR[ClkPol] equals 0 • Data sampled msb first if SICR[SHDIR] equals 0 • Data sampled lsb first if SICR[SHDIR] equals 1
Frame	Frame Sync. In codec mode Frame can be driven from an external codec or can be generated by the internal clock logic. Frame can be programmed as active High or active Low. <ul style="list-style-type: none"> • The frame sync input from the external Codec if SICR[GenClk] equals 0 • The frame sync output to the external Codec if SICR[GenClk] equals 1 • Frame sync is active low if SICR[SyncPol] equals 0 • Frame sync is active high if SICR[SyncPol] equals 1

Table 30-28. PSC Signal Description for Codec Mode (continued)

Signal	Description
BCLK	Bit Clock. In codec mode CLK is: <ul style="list-style-type: none"> • The clock input from the external Codec if SICR[GenClk] equals 0 • The clock output to the external Codec if SICR[GenClk] equals 1
MCLK	Clock output for an external codec

30.5.2.2 Codec Clock and FrameSync Generation

The BCLK and the FrameSync can be inputs that come from an external codec device or they can be internally generated by the PSC and provided as outputs to the external device, under control of bit **SICR[GenClk]**. When the bit **SICR[GenClk]** is set to zero then the BCLK and the FrameSync are inputs. In this case the FrameSync width can be anything from one BCLK period up to the total FrameSync length/period minus one BCLK. If the GenClk bit is set to one, the PSC generates the BCLK and the FrameSync signal. **Figure 30-45** shows how the PSC generates the clocks.

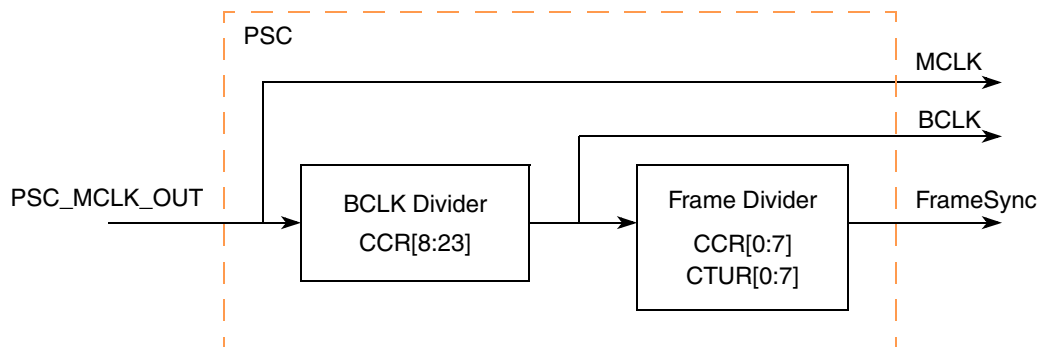


Figure 30-45. Clock Generation Diagram for Codec Mode

The source for the internal clock generation is the PSC_MCLK_OUT provided by the clock module. The PSC provides the MCLK to the external codec divided independently whether the PSC configured as a master (provide BCLK and FrameSync) or as a slave (receive the clock signals).

Each PSC consists of a **CCR** register to generate a BCLK and a FrameSync signal. If the PSC is configured as a master and the MCLK is available, the PSC generates both clock signals independent if the transmitter or receiver is enabled or not. If the PSC configured as SPI master mode the BCLK is only generated if transmitter and receiver are enabled and Tx data are available in the Tx FIFO. The equations below shows the calculation:

Eqn. 30-3

$$BCLK = \frac{MCLK}{BCLKDiv[0:15] + 1}$$

$$FrameSync\ Length = Frame\ SyncDiv[0:7] + 1$$

When the FrameSync is an output, the [CTUR](#) register can program the pulse width. This register defines the number of BCLK cycle during the FrameSync signal is active. The default reset value for this register is 0x00. Therefore, the default FrameSync width is one BCLK. See the calculation below:

Eqn. 30-4

$$\text{Frame sync width} = \text{CTUR}[0:7] + 1$$

30.5.2.3 Transmitting and Receiving in Soft Modem Codec Mode

The PSC supports the full duplex soft modem mode, data is received and transmitted at the same time. To start the full duplex transmission, the Tx and the Rx must be enabled by writing the according value to the [CR](#) register. It's also possible to only use the receiver. For this case, only the Rx enable bit in the [CR](#) register must be set to one. However, it's not possible to use the transmitter without the receiver. To transmit data only, the receiver must be enabled. The received data and the according status and interrupt bits can be ignored.

If the receiver is enabled, the PSC samples data from the receive line after detecting the start of frame condition. The receiver converts the serial data from the Rx line to parallel data words and writes the data to the Rx FIFO. The data word length depends on the programmed word length. If no data exists on the Rx line, the receiver writes zeros to the Rx FIFO until the data word width was reached. The receiver waits until the next start of frame condition is detected. The transmitter converts the parallel data from the Tx FIFO to a serial data stream on the Tx line. If the Tx FIFO is empty during the transmit state, the Tx line is zero. If the last bit of the data word is sent, the transmitter waits until the next start of frame condition is detected.

When [SICR](#)[GenClk] equals 1, the PSC is in master mode and generates the BCLK and the FrameSync signal from the internal clock system, like described in [Section 30.5.2.2, “Codec Clock and FrameSync Generation”](#).

[Figure 30-46](#) shows a codec interface diagram example for soft modem master mode. The different parameters to define the interface are as follows:

- Frame Sync Polarity [SICR](#)[SyncPol]. The leading edge is defined as a rising edge if bit [SICR](#)[SyncPol] equals 1 or a falling edge if [SICR](#)[SyncPol] equals 0
- BCLK Polarity [SICR](#)[ClkPol]. When bit [SICR](#)[ClkPol] equals 0, data is shifted out on the rising edge of bit clock and sampled on the falling edge of BCLK. Otherwise, data is shifted out on the falling edge and sampled on rising edge of bit clock.
- FrameSync width [CTUR](#). Define the number of BCLK while the FrameSync is active.
- FrameSync length [CCR](#)[FrameSyncDiv]. Defines the number of BCLK until the next frame starts.
- Data length [SICR](#)[SIM] defines the data width of the receive and transmit data, 8, 12, 16, 20, 24, or 32 bit per word are possible. In codec 20 and 24 mode, each data sample uses an entire 32-bit longword in the Tx FIFO. The least significant (right-hand) byte is not used. Data should be written to the Tx FIFO four bytes at a time.
- Delay of time slot 1 [SICR](#)[DTS1]. The PSC starts to send a sample at the leading edge of FrameSync [SICR](#)[DTS1] if it equals 0 or 1 bit-clock cycle after the leading edge of FrameSync [SICR](#)[DTS1] if it equals 1.

- Data shift direction **SICR**[SHDIR]. Data shifted out LSB first if **SICR**[SHDIR] equals 1. Otherwise, data shifts out MSB first if **SICR**[SHDIR] equals 0.

In the codec soft modem mode, the PSC sends only one data word per frame.

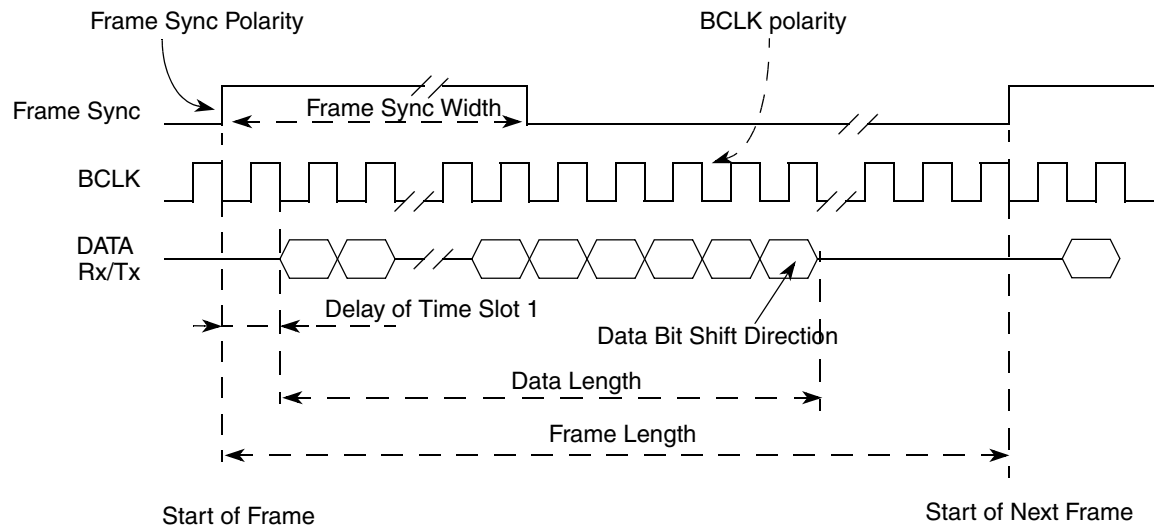


Figure 30-46. Soft Modem Codec Interface Diagram

Table 30-29 shows an example how to configure the PSC1 as:

- PSC in slave mode
- 16-bit soft modem mode
- Data is sampled on the falling edge of BCLK
- FrameSync is low active
- MSB first, transfer starts with leading edge of FrameSync

Table 30-29. 16-Bit Soft Modem Slave Mode

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x02100000	Select the 16 bit Codec mode, msb first, DTS1 = 0, slave mode
CR	0x05	Enable Tx and Rx

Table 30-30 shows an example of how to configure the PSC2 as:

- PSC in Master mode
- 32-bit soft modem mode
- Data is sampled on the rising edge of BCLK
- FrameSync is low active
- The lsb first, transfer starts one cycle after the leading edge of FrameSync
- Set MCLK frequency to 33 MHz
- Set BCLK frequency to 250 kHz
- FrameSync every 35 BCLK
- Set FrameSync width to 3 BCLK

Table 30-30. 32-Bit Soft Modem Master Mode

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x3FA00000	Select the 32bit Codec mode, lsb first, DTS1 = 1, master mode
CCR	0x22830000	select the BCLK and FrameSync frequency
CTUR	0x02	select the FrameSync width
CR	0x05	Enable Tx and Rx

30.5.2.4 Transmitting and Receiving in ESAI Mode (Enhanced Serial Audio Interface)

The ESAI transmission is similar to the soft modem mode. Therefore, the configuration is similar to the description in [Section 30.5.2.3, “Transmitting and Receiving in Soft Modem Codec Mode”](#). The difference is that the ESAI protocol allows to transmit and receive more than one data word per frame. To enable the ESAI mode, the [SICR\[ESAI\]](#) bit must be set. The PSC calculates how many data words the transmitter sends and how much data the receiver expects. [Figure 30-47](#) shows the ESAI transmission diagram.

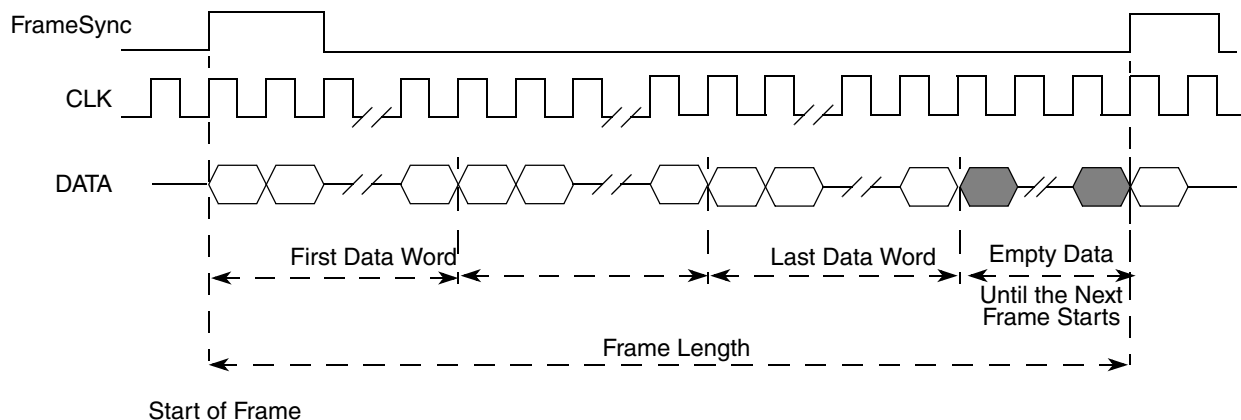


Figure 30-47. ESAI Data Transmission

Table 30-31 shows an example how to configure the PSC1 as ESAI master. For the slave mode, the bit [SICR](#) [GenClk] must be cleared and the configuration of the [CCR](#) register can be ignored. In this configuration example, the PSC sends three data words with 16-bit data in the 52 BCLK frame length. The last four bits in the frame are empty (zero).

- Use PSC1 as ESAI master
- 16-bit data, LSB first
- BCLK frequency 4 MHz
- FrameSync length 52 bit
- Data shifted out on the rising edge of BCLK
- Data transfer starts on FrameSync is active
- FrameSync is active high

Table 30-31. 16-Bit ESAI Master Mode for PSC1

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x12D20000	Select the 16bit Codec ESAI master mode, LSB first, DTS1=0
CCR	0x33030000	Set the FrameSync length (52 bit) and SCKL frequency
CR	0x05	Enable Tx and Rx

30.5.2.5 Transmitting and Receiving in I2S Master Mode

The I2S transmission is similar to the soft modem mode. Therefore, the configuration is as described in [Section 30.5.2.3, “Transmitting and Receiving in Soft Modem Codec Mode”](#). The difference is that during the I2S word transmission the FrameSync signal (LRCK) is stable for the complete data word and is the opposite for the next one. To enable the I2S mode, the [SICR](#)[I2S] bit must be set. The [SICR](#)[SyncPol] bit defines if the frame starts with a low LRCK signal or with a high LRCK signal. If the transmitter detects the start condition, it starts to send the data from the TxFIFO. If the receiver detects a start condition, it starts to write the data from the Rx line to the RxFIFO. The FIFO doesn't provide the ability to mark the data in the FIFO. Therefore, only the order in the FIFO define if the data was received/transmitted during high or low phase of the LRCK. [Figure 30-48](#) shows the I2S transmission diagram.

If an underrun or overflow condition in the FIFO was detected then the receive and the transmit data can't be put to the correct slot of the data transmission. To get in sync with the serial signals, the receiver and the transmitter must be disabled and re-enabled again to start the transmission. At the same time the FIFO must be cleared.

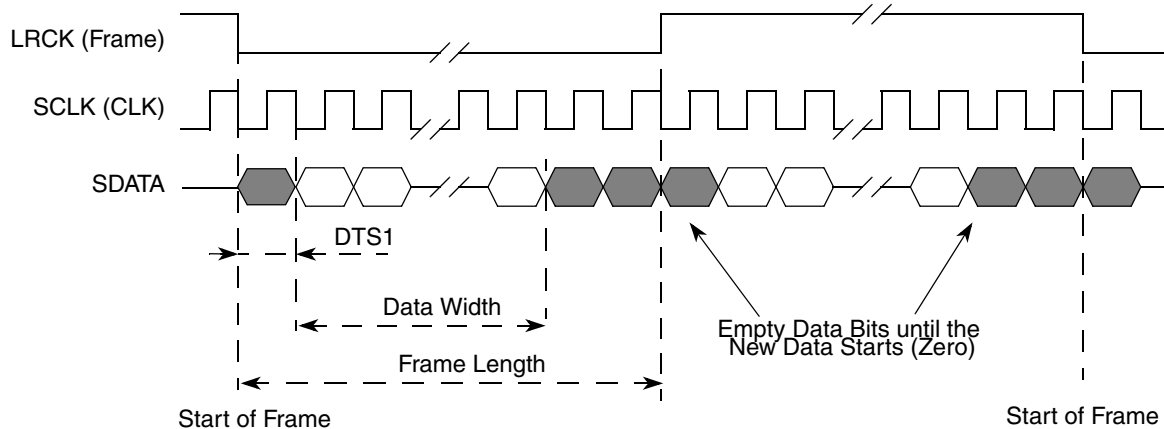


Figure 30-48. I2S-Data Transmission

Table 30-32 shows an example of how to configure the PSC1 as I2S master. For the slave mode the bit **SICR**[GenClk] must be cleared and the configuration of the **CCR** register can be ignored.

- Use PSC1 as I2S master
- 32-bit data, MSB first
- SCLK frequency 1 MHz
- FrameSync width 40 bit
- Data shifted out on the falling edge of SCLK
- Data transfer starts one CLK cycle after the FrameSync is active
- Frame starts with LRCK low

Table 30-32. 32-bit I2S Master Mode for PSC1

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x2FE00000	Select the 32bit Codec I2S master mode, msb first, DTS1 = 1
CCR	0x270F0000	set the FrameSync width (40 bit) and SCKL frequency
CR	0x05	Enable Tx and Rx

30.5.2.6 Transmitting and Receiving in SPI Mode

The PSC supports a full duplex SPI mode. This mode is chosen by setting **SICR**[SPI] equal to 1, only if this bit was set the MSTR, CPOL, CPHA, and UseEOF bits in the **SICR** register take effect. In SPI mode, the **SICR**[SIM] bits must also be set to select the data width. To configure the PSC to act like an SPI master, set **SICR**[MSTR] equal to 1 or set **SICR**[MSTR] equal to 0 to configure the PSC as an SPI slave. When the **SICR**[MSTR] bit is set, **SICR**[GenClk] must also be set to 1 because the PSC is driving the SPI clock line. When **SICR**[MSTR] equals 0, **SICR**[GenClk] must be set to 0 because the external SPI is driving the SCK clock line. The CPOL and CPHA bits in the **SICR** register operate exactly the same way as they do in an SPI, and their values must be the same as the CPOL and CPHA bits in the SPI device communicating with the PSC. The **SICR**[UseEOF] bit has an effect only when **SICR**[MSTR] equals 1 for master mode. If the UseEOF bit is cleared, only one data word (8, 12, 16, 20, 24, or 32 bit width depending

on the [SICR\[SIM\]](#) field) sent before slave select (\overline{SS}) goes high/inactive. When [SICR\[UseEOF\]](#) equals 1, the number of bytes transferred prior to \overline{SS} going high is controlled by the DMA task that fills the Tx FIFO. As the PSC reads bytes out of the Tx FIFO, it holds \overline{SS} low/active until it transmits a byte whose EOF flag is set. In this mode, there is virtually no limit to how many bytes can be sent in one SPI transfer.

The [SICR\[SHDIR\]](#) bit controls the shift direction in SPI mode, as it does in the non-SPI codec modes. The DTS1, ClkPol and SyncPol bits in the [SICR](#) register have no effect in SPI mode.

In SPI master mode, the BCLK (SCK) frequency is generated by dividing the MCLK frequency; see [Section 30.5.2.2, “Codec Clock and FrameSync Generation”](#). In addition to the BCLK generation, the DSCLK delay and the DTL delay must be defined. The DSCLK defines the delay between the \overline{SS} going active and the first BCLK (SCK) clock pulse transition. The DSCLK delay is created by dividing the MCLK frequency. The delay between consecutive transfers is created by dividing the IPS_CLK clock frequency. For more information about the delay generation, see the description of the [CTUR](#), [CTLR](#) and [CCR](#) registers.

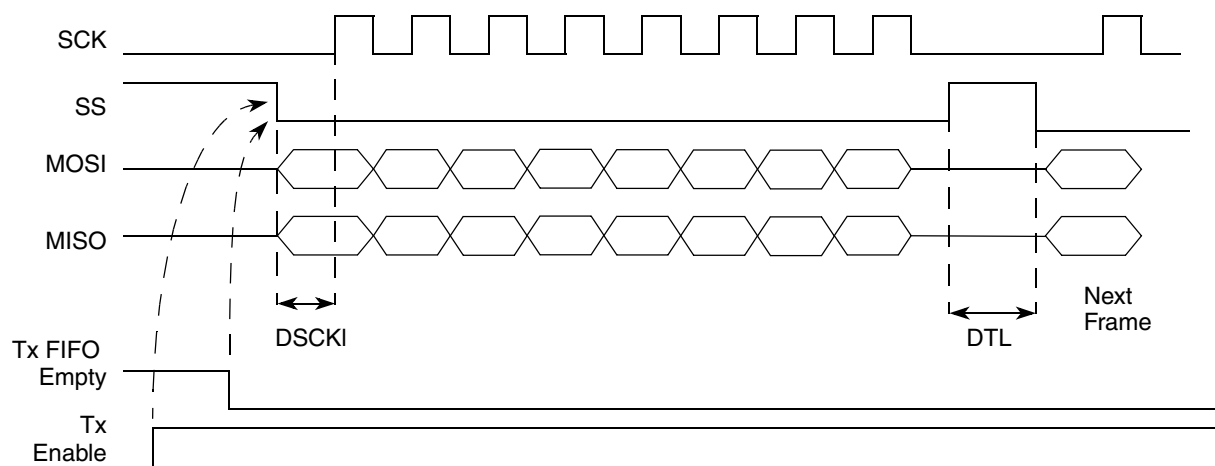
Eqn. 30-5

$$\begin{aligned} \text{DSCKL delay} &= \frac{\text{CCR}[0:7] + 1}{\text{MCLK}} \\ \text{DTL} &= \frac{\text{CT}[0:15] + 2}{\text{IPS_CLK frequency}} + \frac{3}{\text{MCLK frequency}} \end{aligned}$$

where:
 $\text{CT}[0:7] = \text{CTUR}[0:7]$
 $\text{CT}[8:15] = \text{CTLR}[0:7]$

In SPI master mode the PSC controls the serial data transfers. If the Tx FIFO becomes empty (underrun) or the Rx FIFO becomes full (overflow) in the middle of a multi-byte transfer, the PSC keeps the slave select signal low/active and stops the SCK serial clock instead of setting the Tx underrun or Rx overflow status bits. When the Tx FIFO is not empty any more or the Rx FIFO becomes not full, the transfer proceeds.

In SPI slave mode, the MCLK must be running/enabled even though it is not used to generate the serial clock SCK, which is provided by the external master SPI device. The frequency of MCLK is not critical, as long as it is faster than the SCK frequency.



The PSC starts to generate the SCK if the transmitter is enabled and the Tx FIFO is not empty!

Figure 30-49. SPI Parameter

Table 30-33 shows an example of how to configure the PSC3 as SPI master.

- 32-bit data
- Clock is active high, CPOL = 0
- The first SCK edge is issued one half cycle into the data transfer; CPHA = 0
- The msb first
- SCLK frequency 1 MHz
- DSCLK delay = 0.5 μ s
- DTL delay = 2.0 μ s

Table 30-33. 32-bit SPI Master Mode for PSC3

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x0F00C000	Select the 32bit Codec SPI master mode, msb first, CPOL = 0, CPHA = 0
CCR	0x070F	Set the SCK and DSCKL delay
CTUR	0x00	Set the DTL delay 2 us
CTLR	0x84	
CR	0x05	Enable Tx and Rx

Table 30-34 shows an example of how to configure the PSC2 as SPI slave.

- Use PSC2 as SPI slave
- 8-bit data
- Clock is active low, CPOL = 0;
- The first SCK edge is issued at the beginning of the data transfer; CPHA = 1
- The msb first

Table 30-34. 8-bit SPI Slave Mode for PSC2

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x01009000	Select the 8bit Codec SPI slave mode, msb first, CPOL = 0; CPHA = 1
CR	0x05	Enable Tx and Rx

30.5.3 PSC in AC97 Mode

After reset, all PSCs are in UART mode. AC97 mode is chosen by setting the [SICR](#)[SIM] equal to 0x03. The other [SICR](#) field should be initialized at the same time. The important registers to configure the PSC for AC97 mode are:

- [SICR](#) register—select the codec mode
- [CR](#) register—enable or disable receiver and transmitter
- [OP0](#), [OP1](#) register—generate the reset pulse for the external device

30.5.3.1 Block Diagram and Signal Definition for AC97 Mode

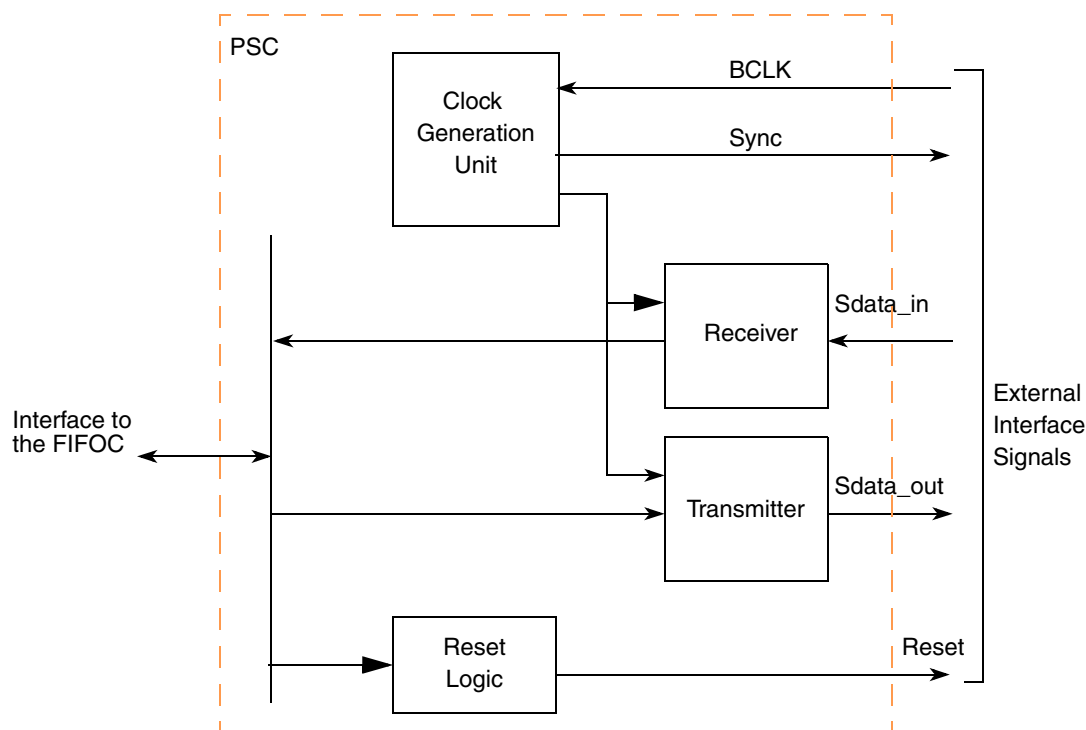


Figure 30-50. PSC AC97 Block Diagram

Figure 30-50 shows the simplified PSC Block Diagram for AC97 mode. The BCLK is an input from the external codec. The PSC divides BCLK by 256 to generate a Frame pulse (Sync) that is high for 16 BCLK cycles. The PSC can only work as an AC97 controller. This means the PSC receives the BCLK from the external AC97 codec and provides the associated frame signal. In AC97 mode, the clock and frame relations are fixed. Therefore, the CCR register and the SICR[GenClk] bit are not used. Table 30-35 shows the pin definition for the AC97 mode, and Figure 30-51 shows an AC97 interface. A general-purpose I/O (GPIO) is used as a reset to the external AC97 device.

Table 30-35. PSC Signal Description for AC97 Mode

Signal	Description
Sdata_out	Transmitter Serial Data Output. Data is shifted out on TxD on the rising edge of the clock signal. Transfers must be specified as msb first.
Sdata_in	Receiver Serial Data Input. Data received on RxD is sampled on the falling edge of the clock signal. Transfers must be specified as msb first.
Sync	In AC97 mode, Sync is the frame sync, or start-of-frame (SOF), output to the external AC97 Controller. In this mode, the AC97 BCLK, which is input on CLK, is divided by 256 to generate the Sync.
BCLK	BCLK. In AC97 mode, CLK must be driven from the external AC97 Controller.
Reset	Reset signal to the external AC97 device

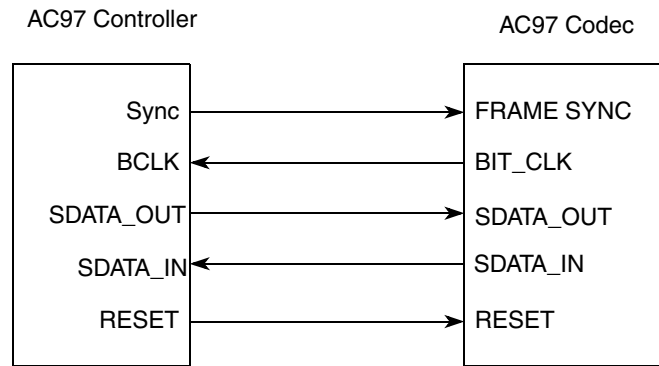


Figure 30-51. PSC – AC97 Interface

Figure 30-52 shows the timing diagram for the AC97 interface. For more AC97 Controller interface information, see the *Audio Codec '97 Component Specification*.

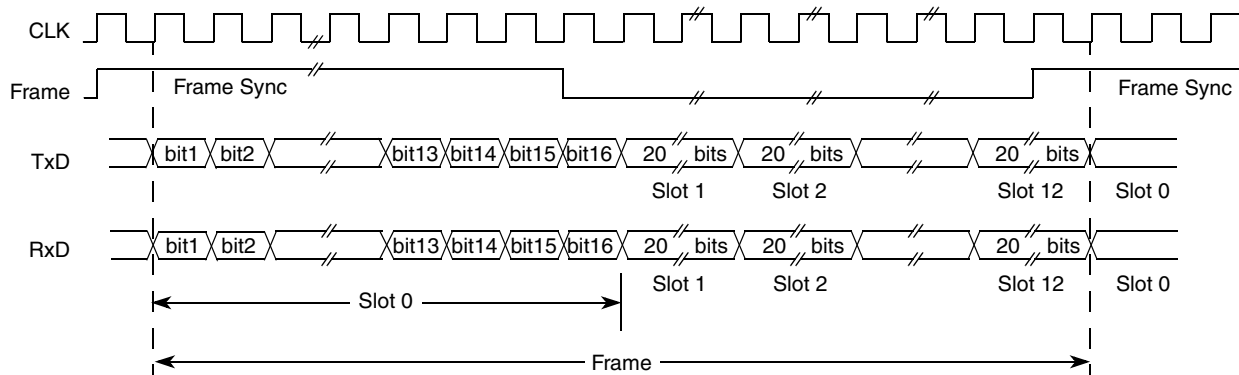


Figure 30-52. Timing Diagram – AC97 Interface

30.5.3.2 Generate a Reset Condition for the External AC97 Codec Device

As described in the AC97 specification, there are three different reset conditions of the AC97 interface.

- Cold AC97 Reset
- Warm AC97 Reset
- Register AC97 Reset

30.5.3.2.1 Cold AC97 Reset

The following sequence generates a reset pulse, on the AC97 Reset line for the external AC97 device. At the same time, the Sync line is forced to low to make sure the external codec doesn't enter the test mode.

- Write 0x02 to the [OP1](#) register; Reset line goes low
- Write 0x02 to the [OP0](#) register; Reset line goes high

30.5.3.2.2 Warm AC97 Reset

To generate a Warm Reset of the AC97 interface, the bit [SICR](#)[AWR] must be set for least at 1 μ s. This asserts the Sync line to high and indicate a Warm AC97 Reset condition to the external AC97 device.

30.5.3.2.3 Register AC97 Reset

To generate a Register Reset, a write access to the reset register of the external device must be performed.

30.5.3.3 External AC97 Codec Test Mode

To bring some external codec devices to test mode, a reset pulse during the high phase of the Sync signal is required. To generate this condition the software must do as follows:

1. set bit [SICR\[AWR\]](#) to force the Sync signal high
2. generate a reset pulse like described in [Section 30.5.3.2.1, “Cold AC97 Reset”](#)

30.5.3.4 AC97 Low-Power Mode

The external AC97 device monitors the first three time slots of each Tx frame to detect the power-down condition for the AC97 digital interface. To put the external device into the Low Power mode, the required bit of the power-down control register of the external device must be set. To do this, the required command data word and the address of the power-down register must be written to the [AC97CMD](#) register. After the write access to the [AC97CMD](#) register the controller sends out the power-down command by:

1. set the first three bits of slot 0, indicating Tx frame and slots 1 and 2 are valid.
2. transmit the address of the power-down control register in Slot 1
3. transmit the command data in Slot 2

Low-Power mode can be left through a Warm or Cold Reset.

30.5.3.5 Transmitting and Receiving in AC97 Mode

The data transmission complies with the standard AC97 one. See [Section 30.5.3.1, “Block Diagram and Signal Definition for AC97 Mode”](#). The AC97 controller is able to generate the data for time slot0,1 and 2 on the transmit side and analyzes received time slot0,1 and 2. The used audio data slots (3 to 12) must be in the FIFOs.

The [RX_SLOTS](#) field in the [AC97Slots](#) register specify the expected Rx data slots. If the received slots don't match this specification, the receiver ignores all data slots from the current frame and sets the [SR\[UNEX_RX_SLOT\]](#) bit. Only the expected and valid tagged data slots are in the Rx FIFO. This functionality guarantees that the software can assign the data in the Rx FIFO to an AC97 slot. Only the order in the Rx FIFO marks the AC97 slot number.

The [Tx_SLOTS](#) field in the [AC97Slots](#) register defines which data slots are sent. All data for these slots must be in Tx FIFO. The transmitter generates the related slot0 tag data. If the Tx FIFO is empty, the transmitter tags the frame as empty. The transmitter sends data if the receiver detects the codec ready state for the current frame. The Tx FIFO contains the specified data words (defined in the [AC97Slots](#) register), and the slot request for the specified slots was active (slot request bit was zero in the previous frame). If the AC97 codec set a slot request to one, the transmitter sends a complete empty frame because the transmitter cannot send part of the required slots without changing the order of the data in the FIFO.

If the software needs to send a command to the AC97 codec, the control register index and the control register write data values must be written to the [AC97CMD](#) register. A write access to any word of this

register triggers the transmitter to send out the register value, at the same time the [SR\[CMD_SEND\]](#) bit was set. The transmitter generates a slot0 tag that marks slot1 and slot2 as valid slot. If the transmitter was able to send out the command data, the [SR\[CMD_SEND\]](#) bit is cleared.

If the receiver detects a valid data in time slot2, the [SR\[DATA_VALID\]](#) bit is set by the receiver. The software can read the received data from the [AC97Data](#) register, synchronous the read access to this register clears the [SR\[DATA_VALID\]](#) bit. If the receive detect an additional command data before the previous data was read out, the [SR\[DATA_OVR\]](#) bit was also set to one. The previous received command data word is lost. A read access to the [AC97Data](#) register clears the [SR\[DATA_VALID\]](#) and [SR\[DATA_OVR\]](#) register. [Table 30-36](#) shows an example how to configure the AC97 controller. In this example, the AC97 controller sends only time slot 3 and slot4 data and expects data for time slot9, 10, 11, and 12 on the receive side. For this purpose, the software must write two data words to the Tx FIFO for one complete AC97 frame and read four data words from the Rx FIFO per frame.

Table 30-36. General Configuration Example AC97 Mode

Register	Value	Setting
CR	0x0A	Disable the Tx and Rx part for configuration if the PSC was enabled earlier.
SICR	0x03010000	Select the enhanced AC97 mode
AC97Slots	0x0300000F	Define the expected receive and transmit slots
IMR	0xFFFF	Select the desired interrupt
CR	0x05	Enable Tx and Rx

30.5.4 Local Loop-Back Mode

[Figure 30-53](#) shows how TxD and Rx are internally connected in local loop-back mode. This mode is for testing the operation of a local PSC module channel by sending data to the transmitter and checking data assembled by the receiver to ensure proper operation. To enable this mode, see register [MR2](#) description.

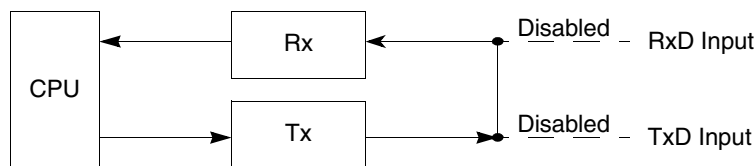


Figure 30-53. Local Loop-Back

Features of this local loop-back mode are:

- Transmitter and CPU-to-receiver communications continue normally.
- Rx input data is ignored.
- TxD data is held marking.
- The receiver is clocked by the transmitter clock.
- Transmitter must be enabled, but the receiver does need not to be enabled.

30.5.5 Remote Loop-Back Mode

In remote loop-back mode, shown in [Figure 30-54](#), the channel automatically transmits received data bit-by-bit on the TxD output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. For this mode, the transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and error status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected. To enable this mode, see register [MR2](#) description.

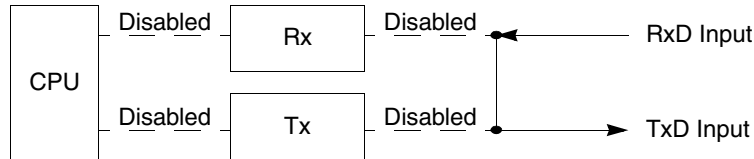


Figure 30-54. Remote Loop-Back

Chapter 31

PSC Centralized FIFO Controller (FIFOC)

31.1 Introduction

The MPC5121e has a centralized FIFO controller that contains data to be transmitted and received data for all 12 PSC modules. For each PSC module, one TX and one RX FIFO space is available. The size of each memory slice is programmable. If the unused FIFO slices are set to zero, another FIFO slice can use this area. To maintain data consistency, multiple slices can not share the same space. The available memory space for all slices together is 32 x1024 (4KB).

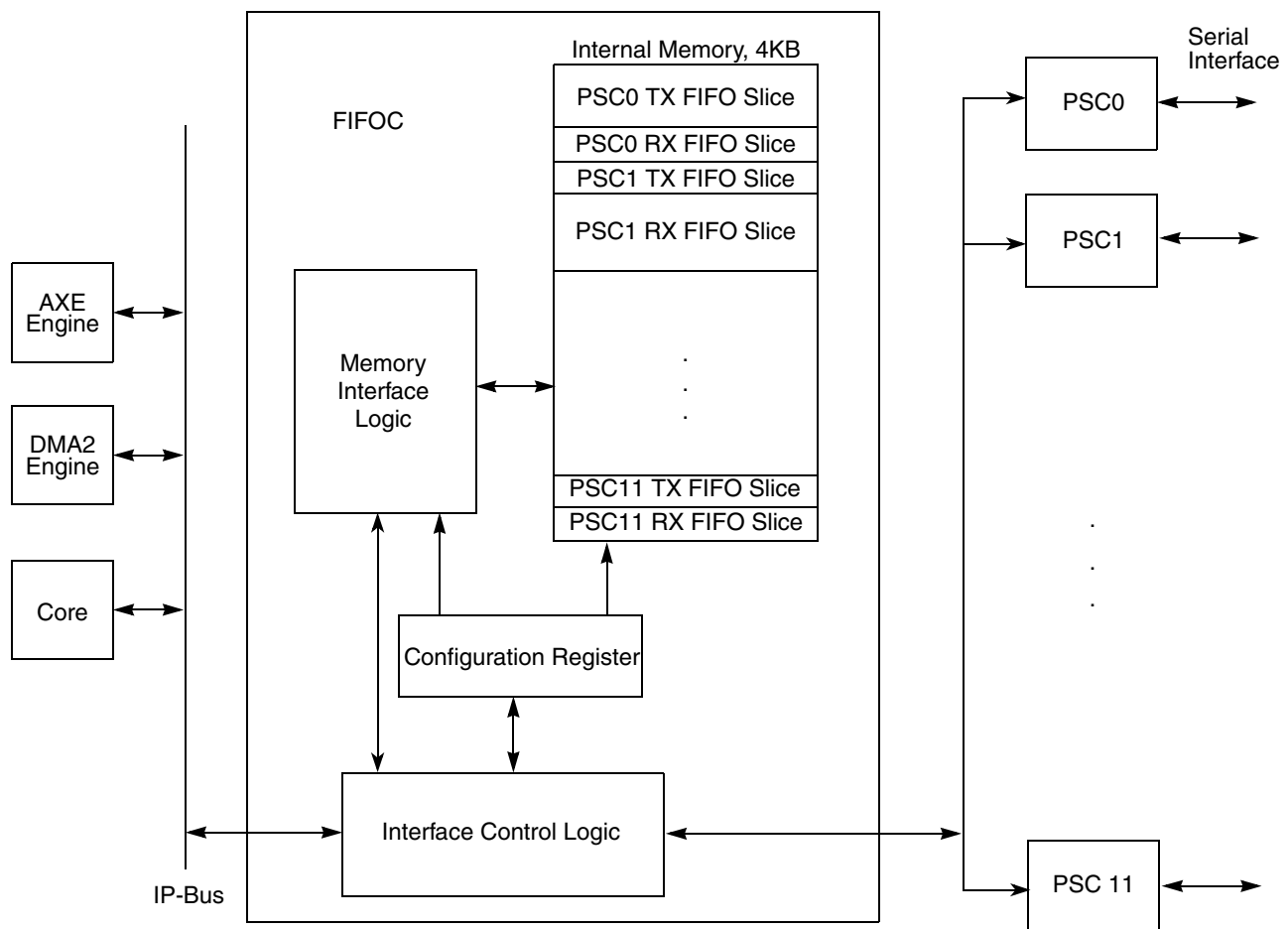


Figure 31-1. FIFOC Overview

31.1.1 Features

Features of the FIFO controller are:

- Independent programmable memory size for transmitter and receiver for each PSC
- All PSCs share the FIFO memory; each PSC FIFO can use as much memory as available depending on the configuration
- Dynamic clock gating function to save power during normal operation
- Independent programmable request signals to the core, DMA2, and AXE engine
- Debug mode to overwrite internal address pointer and write access to ISR register to generate interrupts independent from the interrupt status

31.1.2 Modes of Operation

Behind the normal FIFO controller operation, the FIFOC provides a debug mode that allows the software to manipulate the internal generated pointer register.

The memory map in [Table 31-1](#) shows the complete memory space for all 12 PSCs and the FIFO controller. Access to the PSC memory area passes directly to the 12 PSC modules. The control register area for the TX FIFO (0xn80 - 0xnBC) and the RX FIFO (0xnC0 - 0xnFC) for all 12 PSCs contains the same registers to configure the FIFO area. In this chapter, the character n stands for the supported PSC number 0 to 11. The register area below 0xF00 contains some special register needed to configure the FIFO function independent from the configuration of the individual FIFO areas.

Table 31-1. FIFO Memory Map

Offset		Register Name	Register Width	Access
11..8	7..0			
0x0n (PSC number)	0x00..0x54	Register space for PSC number n	—	—
	0x80	Command register for PSCn TX slice — PSCn_TX_CMD	32	R/W
	0x84	Alarm level for PSCn TX slice —PSCn_TX_ALARM	32	R/W
	0x88	Status register for PSCn TX slice —PSCn_TX_SR	32	R
	0x8C	Interrupt status register for PSCn TX slice —PSCn_TX_ISR	32	R
	0x90	Interrupt mask register for PSCn TX slice —PSCn_TX_IMR	32	R/W
	0x94	FIFO count for PSCn TX slice —PSCn_TX_COUNT	32	R
	0x98	FIFO pointer for PSCn TX slice —PSCn_TX_POINTER	32	R
	0x9C	FIFO size register for PSCn TX slice — PSCn_TX_SIZE	32	R/W
	0xBC	FIFO data register for PSCn TX slice — PSCn_TX_DATA	32	R/W
	0xC0	Command register for PSCn RX slice — PSCn_RX_CMD	32	R/W
	0xC4	Alarm level for PSCn RX slice —PSCn_RX_ALARM	32	R/W
	0xC8	Status register for PSCn RX slice — PSCn_RX_STAT	32	R
	0xCC	Interrupt status register for PSCn RX slice — PSCn_RX_INTSTAT	32	R
	0xD0	Interrupt mask register for PSCn RX slice — PSCn_RX_INTMASK	32	R/W
	0xD4	FIFO count for PSCn RX slice — PSCn_RX_COUNT	32	R
	0xD8	FIFO pointer for PSCn RX slice — PSCn_RX_POINTER	32	R
	0xDC	FIFO size register for PSCn RX slice — PSCn_RX_SIZE	32	R/W
	0xFC	FIFO data register for PSCn RX slice — PSCn_RX_DATA	32	R/W
0xF00		FIFO command	32	R/W
0xF04		FIFO interrupt status	32	R
0xF08		FIFO DMA request	32	R
0xF0C		FIFO AXE request	32	R
0xF10		FIFO debug	32	R/W

31.1.3 Register Descriptions

31.1.3.1 Command Register (CMD)

Offset 0xn80 – Tx

Access: User read/write

Offset 0xnC0 – Rx

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									0							
W								EOF	RE-SET SLICE				AXE_EN	DMA_EN		SLICE_EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 31-2. Command Register (CMD)

Table 31-2. CMD Field Descriptions

Field	Description
EOF	End of Frame 0 Unused 1 Defines the next data word written to the data register as the last word of this frame
RESET SLICE	Reset the FIFO slice. 0 Unused 1 Clears the underrun, overrun, and memory access error bits. Clears all data in the FIFO slice and reset the internal pointer.
AXE_EN	Enable the request to the AXE engine 0 Request to the AXE engine is disabled. 1 Request to the AXE engine is enabled. The FIFO controller generates a request for this slice to AXE engine if this slice is enabled and the current data pointer reaches the alarm level.
DMA_EN	Enable the request to the DMA engine 0 Request to the DMA engine is disabled. 1 Request to the DMA engine is enabled. The FIFO controller generates a request for this slice to DMA engine if this slice is enabled and the current data pointer reaches the alarm level.
SLICE_EN	Enable this Slice of the FIFO 0 FIFO slice is disabled. All interrupt and request lines are cleared. Access to the internal register is possible, but the PSC can't read or write data to the FIFO. 1 FIFO slice is enabled. The FIFO controller provides the data for the transmitter and stores the data from the receiver. If the size of the FIFO slice is zero, it's not possible to enable the FIFO.

31.1.3.2 Alarm Level (ALARM)

Offset 0xn84/0xnC4

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					ALARM LEVEL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 31-3. Alarm Level Register (ALARM)

Table 31-3. ALARM Field Descriptions

Field	Description
ALARM LEVEL	The alarm level defines the number of data bytes in the FIFO when the alarm status appears. Also, the interrupt lines to the core or the request lines to the DMA or AXE engine are asserted if these lines are enabled. For the TX FIFO area, the alarm status appears if the number of data in the TX FIFO is below this alarm level. For the RX FIFO area, the alarm status appears if the number of data in the RX FIFO is above this alarm level. The alarm level and the request lines deassert if the number of data cross this level again.

31.1.3.3 Status Register (SR)

The Status register shows the internal status of the FIFO slice. If an event occurs, the corresponding bit in the SR and in the ISR register is set. If the event disappears, the bit in the SR is also cleared but the bit in the ISR register is active until it is cleared by SW.

Offset 0xn88/0xnC8

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DEBUG MODE								SIZE ZERO	MEM ERROR	DATA READY	ORERR	URERR	ALARM	FULL	EMPTY
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 31-4. Status Register (SR)

Table 31-4. SR Field Descriptions

Field	Description
MEM ERROR	Memory Access Error 0 No Error occurred 1 The access to the data register generates an access error
DATA READY	Data Ready 0 FIFO empty 1 FIFO contains one or more data words
ORERR	Overflow Error 0 No overflow error 1 Overflow error occurred, write access to a full FIFO
URERR	Underrun Error 0 No underrun error 1 Underrun error occurred, read access from an empty FIFO
ALARM	FIFO Alarm 0 The number of data in the FIFO doesn't reach the alarm level 1 The number of data in the FIFO reached the alarm level
FULL	FIFO Full 0 The FIFO is not full 1 The FIFO is full
EMPTY	FIFO Empty 0 The FIFO contains data 1 The FIFO is empty

Offset 0xn88/0xnC8

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DEBUG MODE								SIZE ZERO	MEM ERROR	DATA READY	ORERR	URERR	ALARM	FULL	EMPTY
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 31-4. Status Register (SR)

Table 31-4. SR Field Descriptions (continued)

Field	Description
SIZE ZERO	Size Zero 0 The programmed number of available memory words for this slice inside the Size register is not zero 1 The programmed number of available memory words for this slice inside the Size register is zero
DEBUG MODE	Debug Mode 0 The Debug mode is disabled 1 The Debug mode is enabled

31.1.3.4 Interrupt Status Register (ISR)

Writing one to the interrupt bit clears the interrupt bit.

Offset 0xn8C/0xnCC

Access: User read/1C

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R										MEM ER- ROR	DATA READ Y	ORERR	URERR	ALARM	FULL	EMPTY
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 31-5. Interrupt Status Register (ISR)

Table 31-5. ISR Field Descriptions

Field	Description
MEM ERROR	Memory Access Error. This bit is identical to the MEM error bit in the SR register. If the corresponding bit in the IMR register also set, an interrupt is generated. Writing one to this bit clears the interrupt.
DATA READY	Data Ready. This is identical to the DATA READY bit in the SR register. If the corresponding bit in the IMR register is also set, an interrupt is generated. Writing one to this bit clears the interrupt.
ORERR	Overflow Error. This bit is identical to the ORERR bit in the SR register. If the corresponding bit in the IMR register also set, an interrupt is generated. Writing one to this bit clears the interrupt.
URERR	Underrun Error. This bit is identical to the ORERR bit in the SR register. If the corresponding bit in the IMR register is also set, an interrupt is generated. Writing one to this bit clears the interrupt.
ALARM	FIFO Alarm. This bit is identical to the ALARM bit in the SR register. If the corresponding bit in the IMR register is also set, an interrupt is generated. Writing one to this bit clears the interrupt.
FULL	FIFO Full. This bit is identical to the FULL bit in the SR register. If the corresponding bit in the IMR register is also set, an interrupt is generated. Writing one to this bit clears the interrupt.
EMPTY	FIFO Empty. This bit is identical to the EMPTY bit in the SR register. If the corresponding bit in the IMR register is also set, an interrupt is generated. Writing one to this bit clears the interrupt.

31.1.3.5 Interrupt Mask Register (IMR)

Offset 0xn90/0xnD0

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R										MEM ER- ROR	DATA READ Y	ORERR	URERR	ALARM	FULL	EMPTY
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 31-6. Interrupt Mask Register (IMR)

Table 31-6. IMR Field Descriptions

Field	Description
MEM ERROR	Memory Access Error 0 The memory access error status has no effect on the interrupt line. 1 Enable the interrupt for memory access error.
DATA READY	Data Ready 0 The DATA READY status has no effect on the interrupt line. 1 Enable the interrupt for DATA READY status.
ORERR	Overflow Error 0 The overflow error status has no effect on the interrupt line. 1 Enable the interrupt for overflow error.
URERR	Underflow Error 0 The underflow error status has no effect on the interrupt line. 1 Enable the interrupt for underflow error.
ALARM	FIFO Alarm 0 The FIFO alarm status has no effect on the interrupt line. 1 Enable the interrupt for FIFO alarm.
FULL	FIFO Full 0 The FIFO full status has no effect on the interrupt line. 1 Enable the interrupt for FIFO full.
EMPTY	FIFO Empty 0 The FIFO empty status has no effect on the interrupt line. 1 Enable the interrupt for FIFO empty.

31.1.3.6 Count Register (Count)

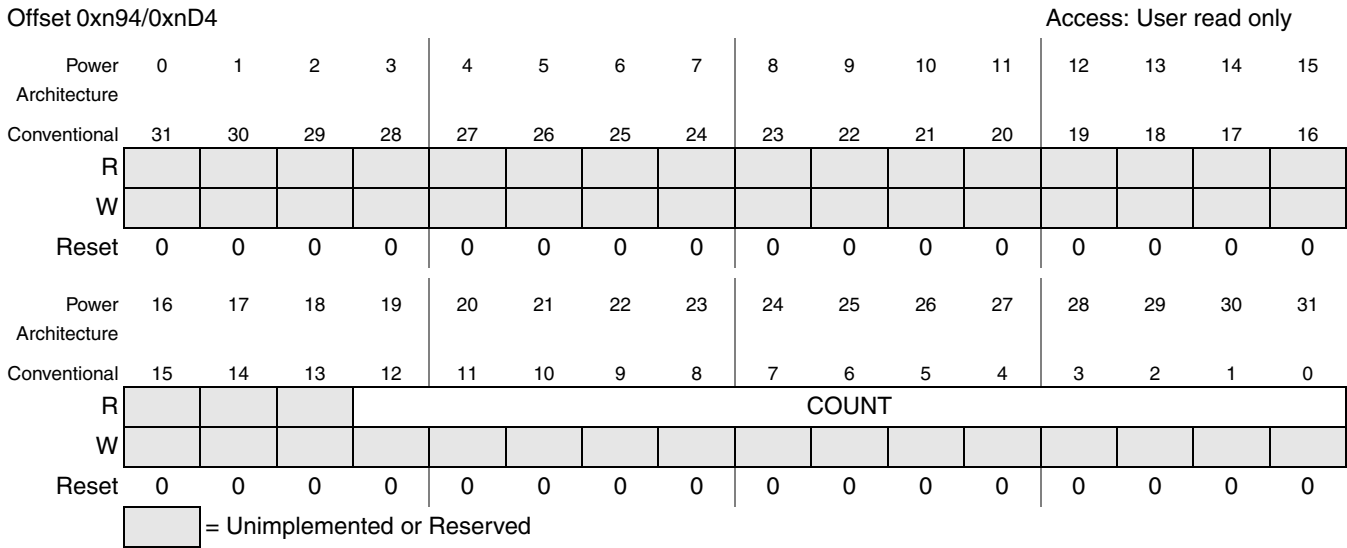


Figure 31-7. Count Register (Count)

Table 31-7. Count Field Descriptions

Field	Description
COUNT	Count. This read only register shows the number of bytes in the FIFO.

31.1.3.7 Pointer Register (PTR)

Offset 0xn98/0xD8

Access: User read only

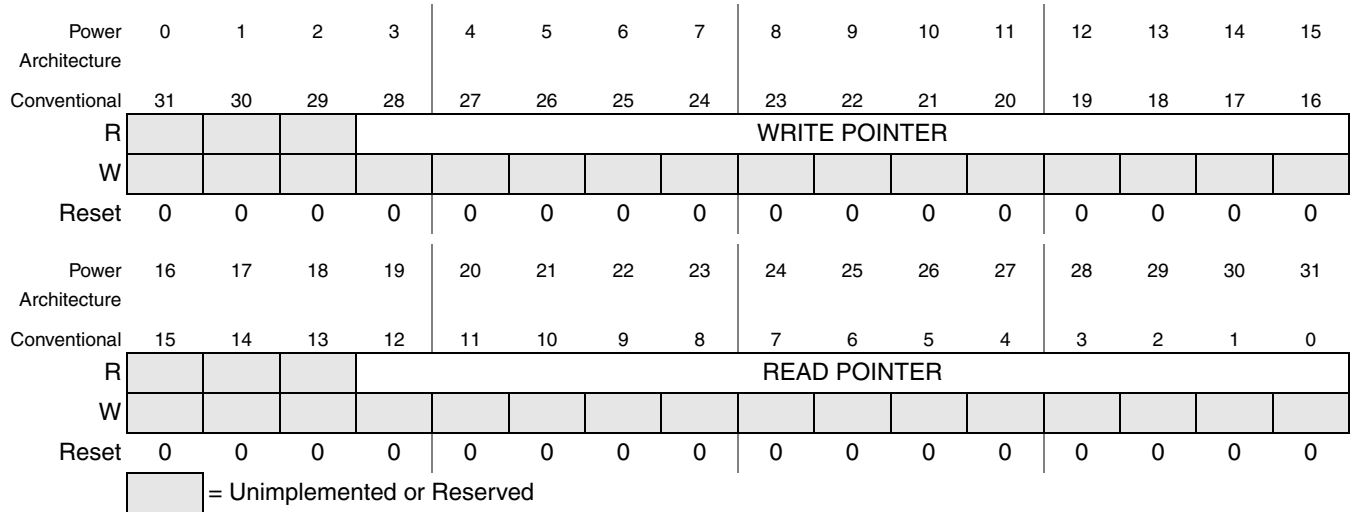


Figure 31-8. Pointer Register (PTR)

Table 31-8. PTR Field Descriptions

Field	Description
WRITE POINTER	Write Pointer. This read only register shows the current write position in the FIFO memory without the offset for the slice number. This register is writeable during debug mode.
READ POINTER	Read Pointer. This read only register shows the current read position in the FIFO memory without the offset for the slice number. This register is writeable during debug mode.

31.1.3.8 FIFO Size Register (Size)

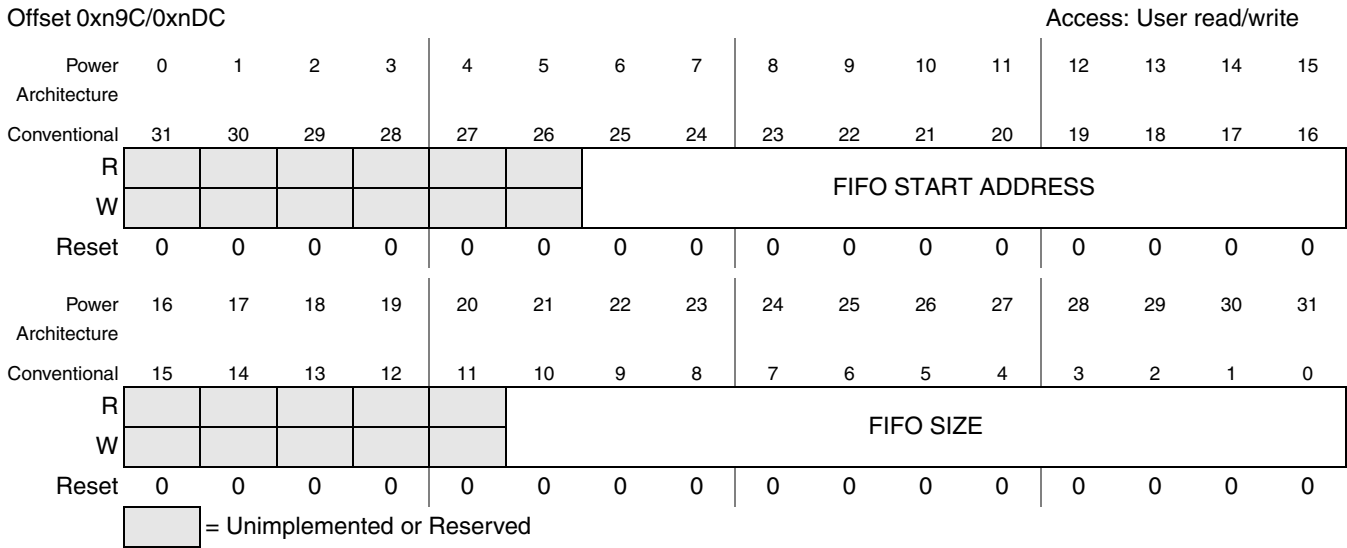


Figure 31-9. FIFO Size Register (Size)

Table 31-9. Size Field Descriptions

Field	Description
FIFO START ADDRESS	Defines the start address for this FIFO slice. The software must take care not to allow overlap in the memory areas to appear.
FIFO SIZE	Defines the size (number of words) of this FIFO slice.

31.1.3.9 Data Register (Data)

Offset 0xnBC/0xnFC

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-10. Data Register (DATA)

Table 31-10. DATA Field Descriptions

Field	Description
DATA	Data Register. Write access to this register writes the data to the FIFO. Read access from this register reads the data from the FIFO. Note: See Section 30.2.1.6, “Rx Buffer Register (RB)” .

31.1.3.10 FIFOC Command Register (FIFOC_CMD)

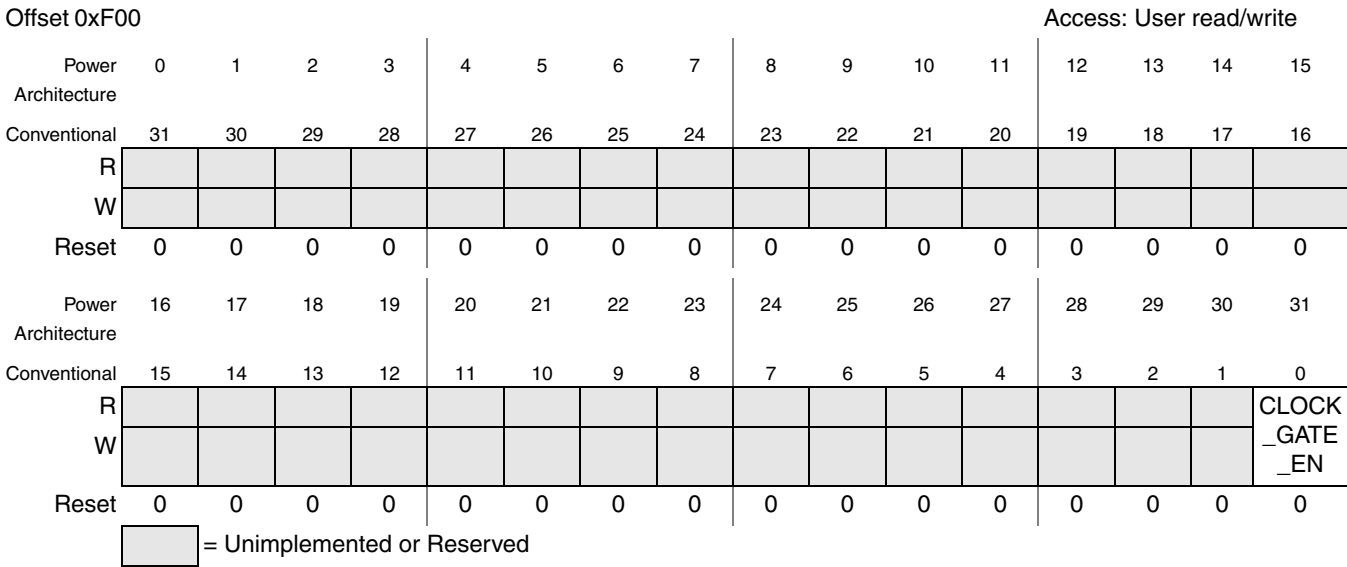


Figure 31-11. FIFOC Command Register (FIFOC_CMD)

Table 31-11. FIFOC_CMD Field Descriptions

Field	Description
CLOCK_GATE_EN	Dynamic Clock Gating Enabled 0 Dynamic clock gating is disabled 1 Dynamic clock gating is enabled, the FIFO gates off the internal clock if no access is available. This behavior increases the number of IP bus wait cycles by one.

31.1.3.11 FIFOC Interrupt Register (FIFOC_INT)

Offset 0xF04

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					FIFOC RX INTERRUPT											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					FIFOC TX INTERRUPT											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 31-12. FIFOC Interrupt Register (FIFOC_INT)

Table 31-12. FIFOC_INT Field Descriptions

Field	Description
FIFOC RX INTERRUPT	FIFOC RX Interrupt. There is one bit per PSC FIFO showing all PSCs with currently pending interrupts. FIFOC RX Interrupt[27:16] = {PSC11...PSC0}
FIFOC TX INTERRUPT	FIFOC TX Interrupt. There is one bit per PSC FIFO showing all PSCs with currently pending interrupts. FIFOC TX Interrupt[11:0] = {PSC11...PSC0}

31.1.3.12 FIFOC DMA Request Register (FIFOC_DMA)

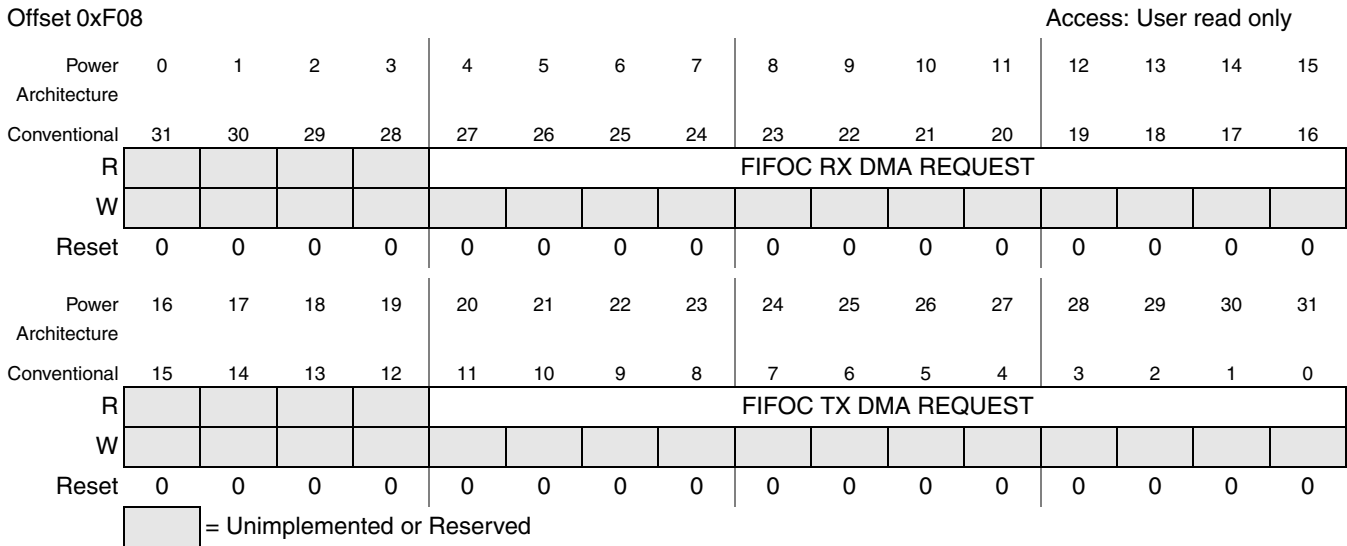


Figure 31-13. FIFOC DMA Request Register (FIFOC_DMA)

Table 31-13. FIFOC_DMA Field Descriptions

Field	Description
FIFOC RX DMA Request	FIFOC RX DMA Request. There is one bit per PSC FIFO showing all PSCs with currently pending requests. FIFOC RX DMA Request[27:16] = {PSC11...PSC0}
FIFOC TX DMA Request	FIFOC TX DMA Request. There is one bit per PSC FIFO showing all PSCs with currently pending requests. FIFOC TX DMA Request[11:0] = {PSC11...PSC0}

31.1.3.13 FIFOC AXE Request Register (FIFOC_AXE)

Offset 0xF0C

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					FIFOC RX AXE REQUEST											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					FIFOC TX AXE REQUEST											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 31-14. FIFOC AXE Request Register (FIFOC_AXE)

Table 31-14. FIFOC_AXE Field Descriptions

Field	Description
FIFOC RX AXE REQUEST	FIFOC RX AXE Request. There is one bit per PSC FIFO showing all PSCs with currently pending requests. FIFOC RX AXE Request[27:16] = {PSC11..PSC0}
FIFOC TX AXE REQUEST	FIFOC TX AXE Request. There is one bit per PSC FIFO showing all PSCs with currently pending requests. FIFOC TX AXE Request[11:0] = {PSC11..PSC0}

31.1.3.14 FIFOC Debug Register (FIFOC_DEBUG)

Offset 0xF10

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R															LOCK	DEBUG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 31-15. FIFOC Debug Register (FIFOC_DEBUG)

Table 31-15. FIFOC_DEBUG Field Descriptions

Field	Description
DEBUG	Debug Mode 0 Normal Operation Mode 1 Debug mode. The Interrupt status register and the pointer register are writeable. The debug bit is only writeable with a 32-bit access to this register during bit[31:16] containing 0x8442.
LOCK	State Machine Lock 0 Normal operation mode 1 Lock the internal machine control state machine. All PSC requests are ignored. Only access from the IP Bus interface is possible. The lock bit is only writeable with a 32-bit access to this register during bit[31:16] containing 0x8442.

31.2 Functional Description

During the functional mode, the FIFOC controller provides the data for all PSC transmitters and stores the data from all PSC receiver. If the FIFO gets a request from the PSC, the required data from the TX FIFO memory is written to the PSC transmit register or the received data is read from the RX register and is stored in the RX FIFO area. If the number of data inside the FIFO register reaches the programmed alarm level, the enabled request lines are asserted to inform the system that the RX data is available or new TX data is required.

Chapter 32

Real Time Clock (RTC)

32.1 Introduction

The real time clock module (RTC) provides a time of day function as well as a calendar function. It also provides a method to bring the MPC5121e from a power-down condition to a fully operational condition. The RTC provides a time base as long as VBAT_RTC power is applied. Parts of RTC module is powered independently of the rest of the MPC5121e device. The RTC has its own internal 32.768 kHz oscillator which is totally independent of the other MPC5121e clock domains. As long as VBAT_RTC is powered, the RTC can be used to cause the MPC5121e to exit any of the power-down modes which include DOZE, NAP, SLEEP, DEEP SLEEP, and HIBERNATION MODE. The wakeup function can be initiated from the internal timer inside the RTC module or from any of six external wakeup pins. An external pin, HIB_MODE, is associated with the RTC and can be used to control the power supplies for the MPC5121e. The RTC controls the HIB_MODE pin in response to any RTC wakeup event, such as a transition on one of the six external wakeup sources or a timeout event in the RTC module.

A block diagram of the RTC is shown in [Figure 32-1](#).

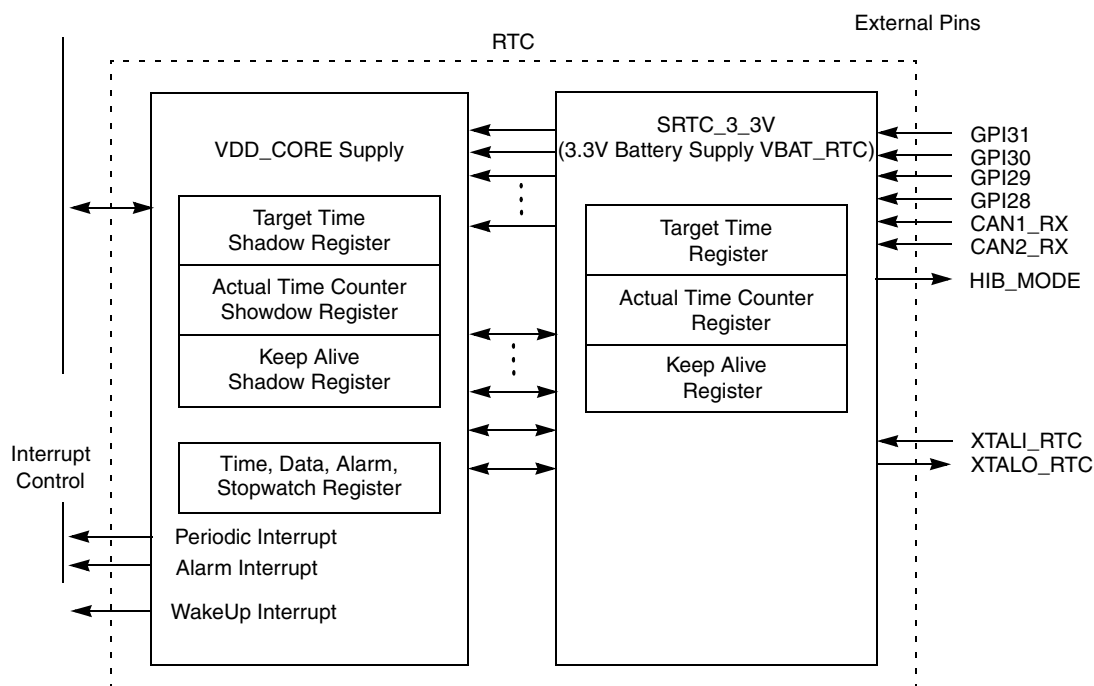


Figure 32-1. RTC Block Diagram

32.1.1 Features

The RTC module provides the following features:

- Secure real time clock (SRTC_3_3V)
 - Secure (not programmable) low-power timebase (actual time count register) that runs continuously as long as VBAT_RTC is powered and the crystal is connected to RTC OSC.
 - HIB_MODE (HIBERNATE) pin used to enable/disable external power regulator
 - Programmable timeout and six external wakeup sources, for exiting HIB_MODE.
 - Programmable wakeup source edge detect
 - Programmable wakeup source inhibitor after the first wakeup source has been detected.
 - Wakeup source bits and four additional bits in RTC keep alive register can be used as an 8-bit breadcrumb register when the MPC5121e is powered off.
 - Loss of power bit, TAMP
- Normal RTC functions under software control
 - minute countdown timer—provides 256-minute capability, slightly over 4 hours
 - programmable alarm—operates on time of day only, not related to calendar
 - periodic interrupts for:
 - 1 second
 - 1 minute
 - 1 day—operates only at midnight rollover
 - calendar features:
 - weekday
 - date
 - year
- Crystal support (32.768 kHz only)

The RTC is divided into two major sections. One section contains the target time shadow register, actual time counter shadow register, keep alive shadow register and the time, date, alarm, and stopwatch registers. This section is powered by the VDD_CORE power supply. These registers retain their contents when powered by the VDD_CORE core power supply. If the VDD_CORE power supply is turned off, these registers lose their contents.

To provide for the restoration of the date and time after all power is lost, except VBAT_RTC, the system software needs to record a value of the actual time count register that corresponds with the contents of the RTC current time register and current date register. These values must be stored in non-volatile memory. Then, after power is restored, software must be used to deduce the current date and time based on the current count of the actual time count register.

The time, date and stopwatch registers are updated by the 1-second clock that is derived from the RTC 32.768 kHz oscillator. When the MPC5121e enters the deep sleep mode, the contents of these registers are maintained and updated as long as the VDD_CORE supply is powered. That is, these registers are updated

by the 1-second clock derived from the RTC 32.768 kHz oscillator; however, when in the deep sleep mode, the real time clock cannot generate a wakeup interrupt based on an alarm time to cause the MPC5121e to exit the deep sleep mode. The deep sleep mode can only be exited in response to an external RTC wakeup pin or when the actual time count register is greater than or equal to the target time register.

The second section of the RTC contains the target time register, actual time counter register, and the keep alive register. This section of the RTC is powered by the VBAT_RTC supply. As long as power is applied to the VBAT_RTC pin, the actual time counter register continues to increment at a 1 Hz rate.

When the CPU reads the value of the actual time count register, target time register or the keep alive register, the value in the respective shadow register is actually fetched. The contents of the actual time count register, target time register or the keep alive register are transferred to their respective shadow registers on each cycle of the 1-second clock which increments the actual time count register.

For predictable operation, consecutive writes to the actual time count shadow register, the target time shadow register or the keep alive shadow register must be separated by at least three clock periods of the 32.768 kHz RTC clock.

32.2 External Signal Descriptions

Table 32-1. External Pins Signal Properties

Name	Function	I/O	Reset
XTALI_RTC	32kHz crystal input	I	—
XTALO_RTC	32kHz crystal output	O	—
$\overline{\text{HIB_MODE}}$	Power regulator disable (See Table 32-14 and Table 32-16)	O	1
GPI31	General purpose input	I	—
GPI30	General purpose input	I	—
GPI29	General purpose input	I	—
GPI28	General purpose input	I	—
CAN1_RX	CAN Receive input	I	—
CAN2_RX	CAN Receive input	I	—

32.3 Memory Map and Register Definition

32.3.1 Memory Map

The MPC5121e RTC uses twelve 32-bit registers. Hyperlinks to the RTC registers are provided in [Table 32-2](#).

Table 32-2. RTC Memory Map

Offset or Address	Register	Access	Section/Page
General Registers			
0x00	RTC time set register – TSR	R/W	32.3.2.1/32-5
0x04	RTC date set register – DSR	R/W	32.3.2.2/32-9
0x08	RTC new year and stopwatch register – NY_STP	R/W	32.3.2.3/32-13
0x0C	RTC alarm and interrupt enable register – ALM_IE	R/W	32.3.2.4/32-14
0x10	RTC current time register – CTR	R	32.3.2.5/32-16
0x14	RTC current date register – CDR	R	32.3.2.6/32-17
0x18	RTC alarm and stopwatch interrupt register – ALM_STP_INT	R	32.3.2.7/32-18
0x1C	RTC periodic interrupt and bus error register – PI_BE	R	32.3.2.8/32-19
0x20	RTC target time register – TTR	R/W	32.3.2.9/32-20
0x24	RTC actual time counter – ATC	R	32.3.2.10/32-21
0x28	RTC keep alive register – KAR	R/W	32.3.2.11/32-22

NOTE

All registers beside RTC target time, RTC actual time counter, and RTC keep alive registers lose content when VDD_CORE power is no longer supplied. The three not influenced registers are powered by VBAT_RTC.

32.3.2 Register Descriptions

32.3.2.1 RTC Time Set Register

This register is used to program the RTC current time register ([Section 32.3.2.5, “RTC Current Time Register”](#)). This register does not reflect the running time value.

Offset 0x00Access: User read/write

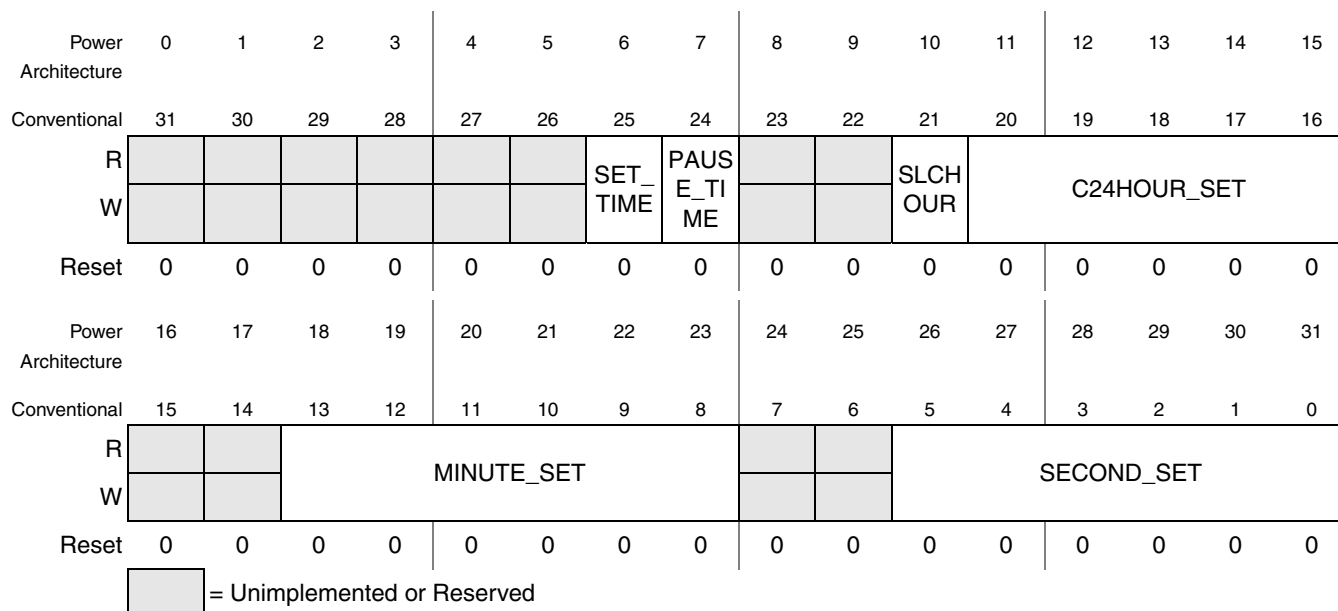


Figure 32-2. RTC Time Set Register
(The register is repeated for reference.)

Table 32-3. RTC Time Set Register Field Descriptions

Field	Description
SET_TIME	<p>A software sequence using the pause time bit in conjunction with the set_time bit must be followed to load new values into the RTC current time register (Section 32.3.2.5, “RTC Current Time Register”). The SET_TIME bit cannot be set to 1 unless the PAUSE_TIME bit is also set to 1.</p> <p>The proper software sequence is:</p> <ul style="list-style-type: none"> • Step 1. Write register with C24hour_set, minute_set, and second_set fields set to the desired values and with pause_time = 1 and set_time = 0 • Step 2. Write register with C24hour_set, minute_set, and second_set fields set to the desired values and with pause_time = 1 and set_time = 1 • Step 3. Write register with C24hour_set, minute_set, and second_set fields set to the desired values and with pause_time = 1 and set_time = 0 • Step 4. Write register with C24hour_set, minute_set, and second_set fields set to the desired values and with pause_time = 0 and set_time = 0 • At completion of step 4, RTC current time register is updated with the new time. <p>The Chour_set, minute_set, and the second set fields should remain consistent throughout the four steps (i.e., at the desired new time values).</p> <p>The hour, minute and second fields of the RTC current time register are all set at the completion of step 4 of the above sequence.</p> <p>It is important to use LOAD and STORE operations to access this register. Do not use read-modify-write instructions such as AND or OR to modify this register.</p>
PAUSE_TIME	Used with set_time above to perform time update. Must be zero for normal operation.

Figure 32-3.

Real Time Clock (RTC)

Offset 0x00Access: User read/write

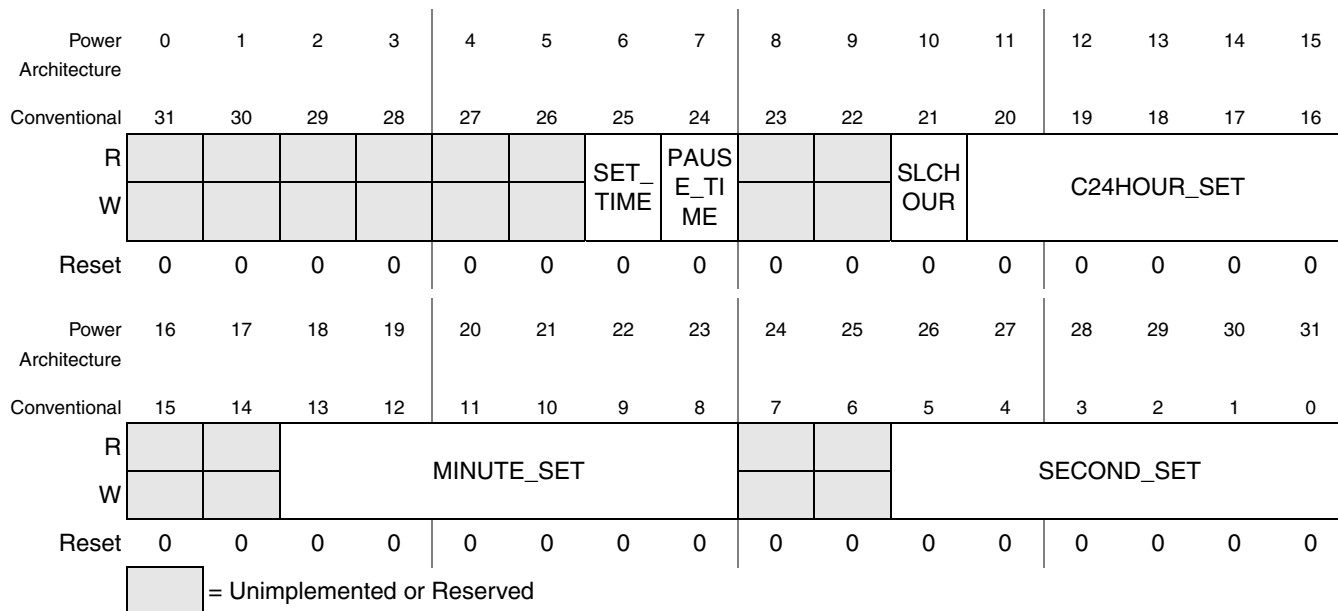


Figure 32-2. RTC Time Set Register
(The register is repeated for reference.)

Table 32-3. RTC Time Set Register Field Descriptions (continued)

Field	Description
SLCHOUR	This bit determines the hour output format. 0 24-hour format 1 12-hour format with AM/PM This bit does not affect time set procedure, it only affects how the hour status field is presented. The new hour format is updated immediately in the current time register.
C24HOUR_SET	Hour in 24-hour format written in RTC current time register after successful state machine transition by set_time and pause_time bits. This field is always written with 24-Hour format, it is not affected by SlctHour bit above.
MINUTE_SET	Minute written in RTC current time register after successful state machine transition by set_time and pause_time bits.
SECOND_SET	Second written in RTC current time register after successful state machine transition by set_time and pause_time bits.

Figure 32-3.

32.3.2.2 RTC Date Set Register

This register is used to program the RTC current date register ([Section 32.3.2.6, “RTC Current Date Register”](#)). The RTC date set register does not reflect the running date value. Notice that the value in year_set field of the RTC new year and stopwatch register ([Section 32.3.2.3, “RTC New Year and Stopwatch Register”](#)) is also loaded into the current date register after the proper software sequence using the set_date and pause_date bits.

Offset 0x04Access: User read/write

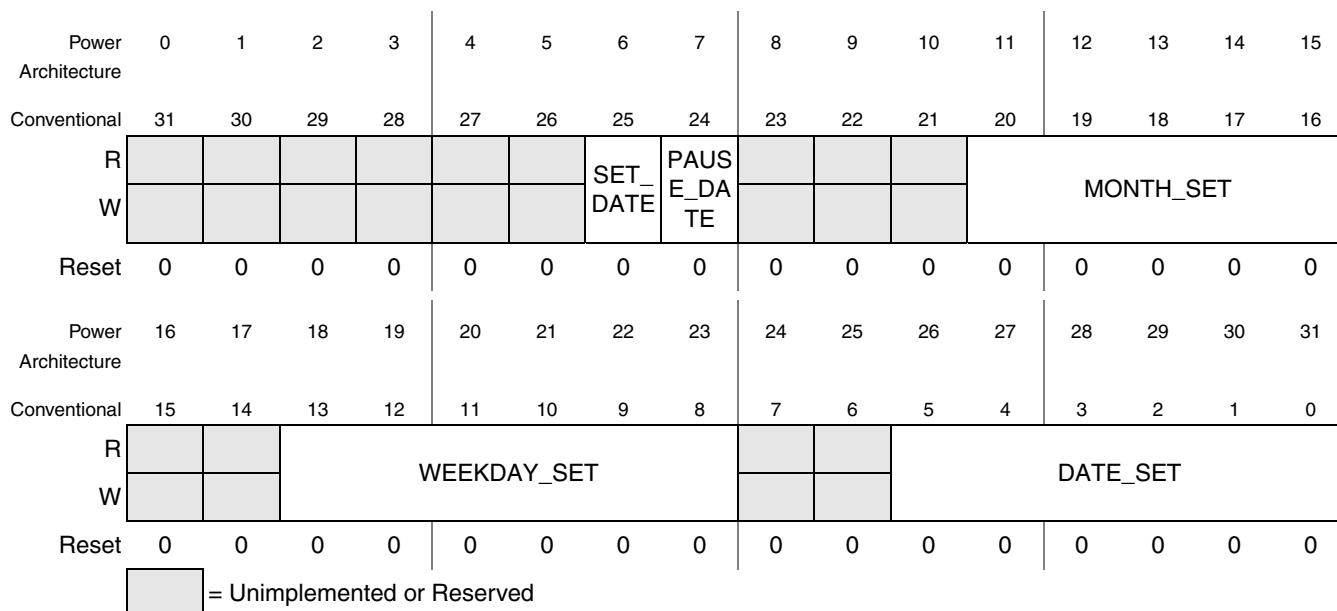


Figure 32-4. RTC Date Set Register
(The register is repeated for reference.)

Table 32-4. RTC Date Set Register Field Descriptions

Field	Description
SET_DATE	<p>A software sequence using the pause_date bit in conjunction with the set_date bit must be followed to load new values into the RTC current time register (Section 32.3.2.5, “RTC Current Time Register”). The SET_DATE bit cannot be set to 1 unless the PAUSE_DATE bit is also set to 1.</p> <p>The proper software sequence is:</p> <ul style="list-style-type: none"> • Step 1. Write register with month_set, weekday_set, date_set and year_set fields set to the desired values and with pause_date = 1 and set_date = 0 • Step 2. Write register with month_set, weekday_set, date_set and year_set fields set to the desired values and with pause_date = 1 and set_date = 1 • Step 3. Write register with Month_set, weekday_set, date_set and year_set fields set to the desired values and with pause_date = 1 and set_date = 0 • Step 4. Write register with Month_set, weekday_set, date_set and year_set fields set to the desired values and with pause_date = 0 and set_date = 0 • At completion of Step 4, RTC Current date register is updated with the new time. <p>The month_set, weekday_set, and the date_set fields should remain consistent throughout the four steps (i.e., at the desired new time values).</p> <p>The year_set field is located in the RTC new year and stopwatch register. The month, weekday, date and year fields are all set at the completion of step 4 of the above sequence.</p> <p>It is important to use LOAD and STORE operations to access this register. Do not use read-modify-write instructions such as AND or OR to modify this register.</p>
PAUSE_DATE	Used with set_time above to perform time update. Must be zero for normal operation.
Month_set	New month written in RTC current date register after successful state machine transition by set_date and pause_date bits. Only the lower 4 bits of this field are used.

Figure 32-5.

Offset 0x04Access: User read/write

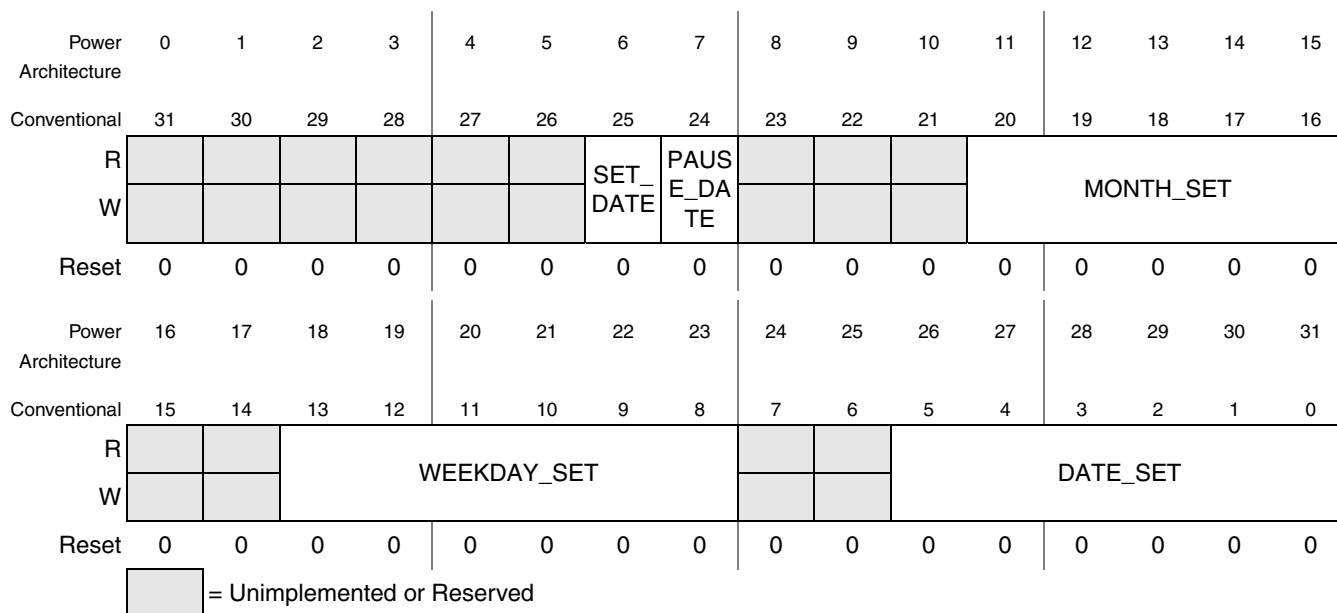


Figure 32-4. RTC Date Set Register
 (The register is repeated for reference.)

Table 32-4. RTC Date Set Register Field Descriptions (continued)

Field	Description
WEEKDAY_SET	New weekday written in RTC current date register after state machine transition by set_date and pause_date bits. 1 = Monday; 7 = Sunday. Only the lower 3 bits of this field are used. If zero is written (not recommended), the weekday is set to zero until the end of the day. The weekday is then incremented to 1.
DATE_SET	<p>New date (1-31) is written in RTC current date register after state machine transition by set_date and pause_date bits. Only the lower 5 bits of this field are used.</p> <p>A date of 1 indicates the first day of the month, 2 indicates the second day of the month, 31 indicates the last day of the month of January.</p> <p>If zero is written (not recommend), the day is set to zero until the end of the day. The date is then incremented to 1.</p> <p>Year_set in the following register is also part of the date set function.</p>

Figure 32-5.

32.3.2.3 RTC New Year and Stopwatch Register

Offset 0x08 Access: User read/write

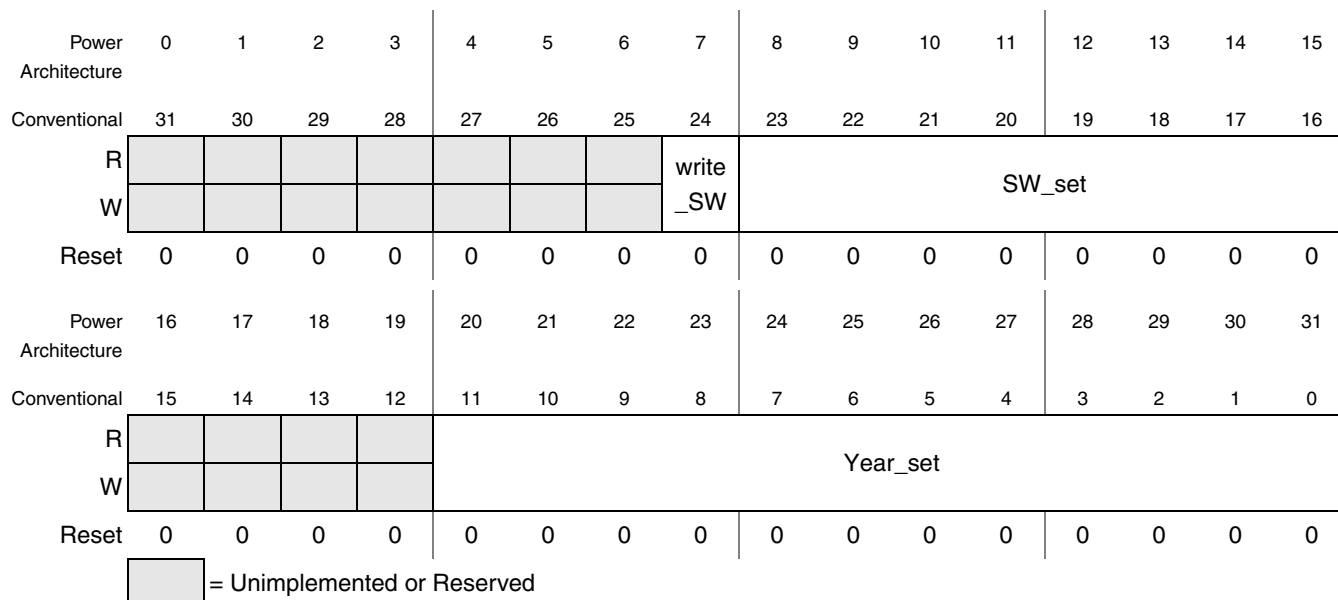


Figure 32-6. RTC New Year and Stopwatch Register

Table 32-5. RTC New Year and Stopwatch Register Field Descriptions

Field	Description
write_SW	Typical stopwatch operation is to write an initial value into the 8-bit wide sw_set field and assert the write_sw bit. The write_sw bit is immediately auto cleared, but it triggers the stopwatch minute countdown to begin.
SW_set	Number of minutes to be written into stopwatch. Max is 255, a little over 4 hours.
Year_set	New year value to be written to the RTC current date register after successful state machine transition by set_date and pause_date bits. This is part of date set function in the previous register.

Figure 32-7.

32.3.2.4 RTC Alarm and Interrupt Enable Register

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R								Alm_enable					Alm_24H_set			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			Alm_Min_set										$\overline{\text{MPE}}$	IntEn_day	IntEn_min	IntEn_sec
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

 = Unimplemented or Reserved

Figure 32-8. RTC Alarm and Interrupt Enable Register

Table 32-6. RTC Alarm and Interrupt Enable Register Field Descriptions

Field	Description
Alm_enable	Alarm enable bit for once-a-day alarm 1 Alarm status/interrupt operation is enabled 0 Alarm setting is not compared to time of day
Alm_24H_set	Hour setting (in 24 hour format) to be compared to time of day for the purpose of generating alarm status/interrupt. Can be written at any time.
Alm_Min_set	Minute setting to be compared to time of day for the purpose of generating alarm status/interrupt. Can be written at any time.
$\overline{\text{MPE}}$	Master periodic enable – Must be written low after reset to allow periodic interrupts. 0 Allow periodic interrupts 1 Do not allow periodic interrupts
IntEn_day	Enable bit of periodic interrupts at midnight rollover.
IntEn_min	Enable bit of periodic interrupts at minute rollover.
IntEn_sec	Enable bit of periodic interrupts at second rollover.

Figure 32-9.

NOTE

The Alarm function generates an interrupt based on a comparison to the hour and minute fields of the RTC current time register, shown in [Figure 32-10](#). Thus, if the alarm function is enabled and left enabled, an alarm is generated each day at a particular time. It is important not to confuse the alarm interrupt with the midnight rollover interrupt. Either one can cause an interrupt once each 24 hours.

When in deep sleep, the alarm function cannot be used to exit the deep sleep mode.

32.3.2.5 RTC Current Time Register

Offset 0x10Access: User read only

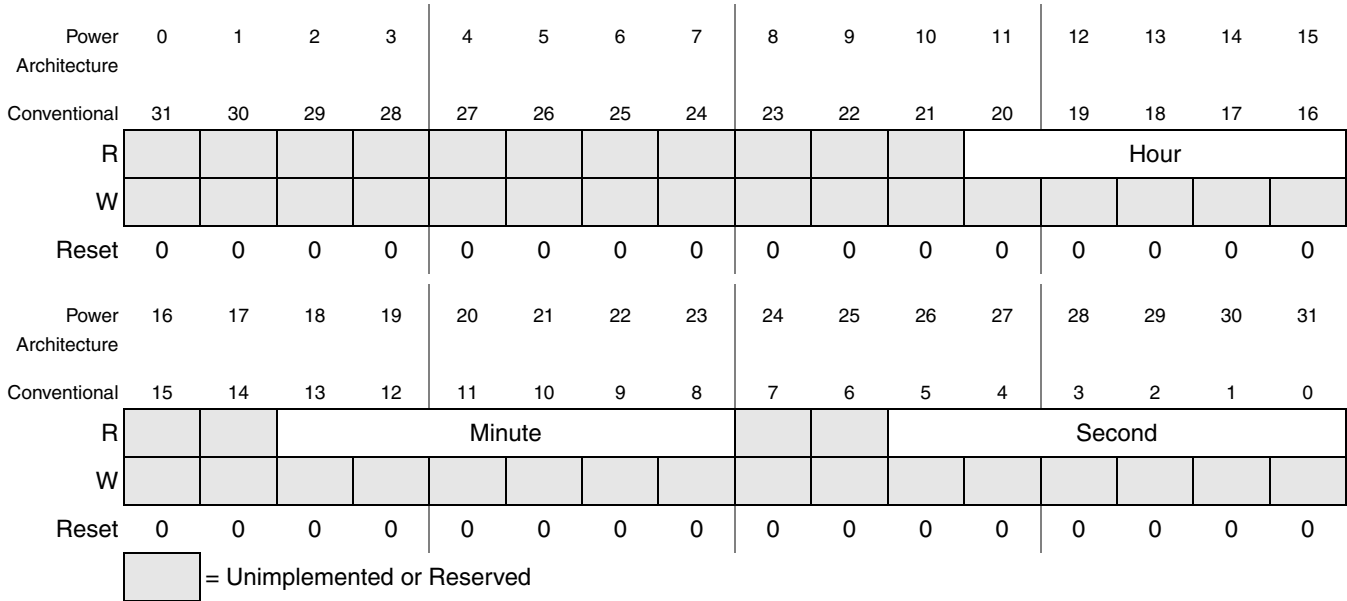


Figure 32-10. RTC Current Time Register

Table 32-7. RTC Current Time Register Field Descriptions

Field	Description
Hour	Hour format can be either 24-hour or 12-hour with AM/PM. If 24-hour format is selected (SlctHour low in Hour[0]), the whole 5-bit hour field designates current time in 24-hour format. If 12-hour format is selected (SlctHour high in Hour[0]), the MSB of hour field indicates: Hour[0]=0: AM Hour[0]=1: PM and Hour[1:4] designates current time in 12-hour format.
Minute	Shows minutes in current time.
Second	Shows seconds in current time.

Figure 32-11.

32.3.2.6 RTC Current Date Register

Offset 0x14 Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R					Month				Weekday				Date			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					Year											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 32-12. RTC Current Date Register

Table 32-8. RTC Current Date Register Field Descriptions

Field	Description
Month	Shows current month. (1 = January; 12 = December)
Weekday	Indicates day of week. (Monday = 1, Sunday = 7)
Date	Shows current date. Calendar feature is implemented, therefore, day rollover at the end of month including February (and Leap Years) is automatic.
Year	Shows current year. Max is 4052.

Figure 32-13.

32.3.2.7 RTC Alarm and Stopwatch Interrupt Register

Offset 0x18 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R								int_alm								int_SW
W								W1C								W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								Alm_status	SW_min							
W								W1C								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 32-14. RTC Alarm and Stopwatch Interrupt Register

Table 32-9. RTC Alarm and Stopwatch Interrupt Register Field Descriptions

Field	Description
Int_alm	Status bit indicating that enabled once-a-day alarm has occurred (active high). Alarm interrupt has been activated. This bit and the Interrupt are cleared by writing 1 to this bit position. A stopwatch interrupt, if also active, must be cleared before the interrupt signal to the CPU is negated.
Int_SW	Status bit indicating that stopwatch expiration has occurred (active high). Stopwatch interrupt has been activated. This bit and the Interrupt are cleared by writing 1 to this bit position. An alarm interrupt, if also active, must be cleared before the interrupt signal to the CPU is negated.
Alm_status	Status bit indicating that once-a-day alarm has occurred. Same as int_alm bit above except that clearing this bit does not clear the interrupt.
SW_min	Minutes remaining in stopwatch.

Figure 32-15.

32.3.2.8 RTC Periodic Interrupt and Bus Error Register

Offset 0x1CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R								Bus_error_1								int_day
W								W1C								W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								int_min								int_sec
W								W1C								W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 32-16. RTC Periodic Interrupt and Bus Error Register

Table 32-10. RTC Periodic Interrupt and Bus Error Register Field Descriptions

Field	Description
Bus_error_1	Internal status register—If high, indicates software has attempted a write access to a read-only register in this module. No actual register contents are corrupted and no interrupt is generated if this happens. Cleared by writing 1 to this bit position.
Int_day	Periodic interrupt at midnight. High indicates interrupt has occurred. Cleared by writing 1 to this bit position.
Int_min	Periodic interrupt at each minute rollover. High indicates interrupt has occurred. Cleared by writing 1 to this bit position.
Int_sec	Periodic interrupt at each second rollover. High indicates interrupt has occurred. Cleared by writing 1 to this bit position.

Figure 32-17.

32.3.2.9 RTC Target Time Register

This register is used to program a time to hibernate, in seconds, based on value in the RTC current time register. When the RTC ATC register is greater than or equal to the RTC time target register, the $\overline{\text{HIB_MODE}}$ signal asserts to a logic 1.

Offset 0x20 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TTR[0:15]															
W																
Reset ¹	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TTR[16:31]															
W																
Reset ¹	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. This register is reset only when VBAT_RTC is connected. After that, it is up to software to make sure it is updated correctly.

Figure 32-18. RTC Target Time Register

Table 32-11. RTC Target Time Register Field Descriptions

Field	Description
TTR	The number, in seconds, to generate the $\overline{\text{HIB_MODE}}$ signal. When TTR = 0x00000000, entering Hibernation or Deep Sleep mode is not possible. If it is desired to use one of the wakeup sources to wakeup (Section 32.3.2.11, “RTC Keep Alive Register”), then write TTR to 0xFFFFFFFF. This allows to assert the $\overline{\text{HIB_MODE}}$ signal and to enter Deep Sleep mode, when non of the external wake up sources (GPI[31:28] or CAN[1:2]_RX) prevent this.

Figure 32-19.

NOTE

Because of the uncertainty of the time that the actual time counter increments, it is recommended that the target time register be written with a value that is two counts greater than the current value of the ATC register.

32.3.2.10 RTC Actual Time Counter

This register is used to read the elapsed time, in seconds, since VBAT_RTC was last connected.

Offset 0x24 Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ATC[0:15]															
W																
Reset ¹	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ATC[16:31]															
W																
Reset ¹	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

1. This register is reset only when VBAT_RTC is connected. Software **cannot** modify this register.

 = Unimplemented or Reserved

Figure 32-20. RTC Actual Time Counter Register

Table 32-12. RTC Actual Time Counter Register Field Descriptions

Field	Description
ATC	The number, in seconds, since VBAT_RTC was last connected.

Figure 32-21.

NOTE

This register is set to 0xFFFF FFFE when VBAT_RTC is applied. If VBAT_RTC is removed and later connected, the register is again set to 0xFFFF FFFE.

The contents of this register cannot be modified by software. After VBAT_RTC is connected, this register increments at a 1 Hz rate starting from 0xFFFF FFFE.

32.3.2.11 RTC Keep Alive Register

This register contains a unique 9-bit breadcrumb register (BC_[0-8]), which can also function as a 5-bit wakeup source register (GPI[30:28] and CAN[1:2]_RX) and a 4-bit breadcrumb register (BC_[5-8]). Each bit that can function as both a wakeup source and as a breadcrumb bit. This register also contains a dedicated wakeup source register (GPI31).

Breadcrumb bits are generally used to hold status words or information that is user defined. As long as V_{BAT_RTC} is powered, this register retains its contents. One purpose for the breadcrumb bits might be to hold user defined information about what process should be executed upon exiting the HIBERNATE mode.

NOTE

When writing to the RTC keep alive register, do not modify any of the WU_SRC_[1:5]_EN bits in the same instruction that modifies any of the WU_SRC_[1:5]_LVL bits. Use one instruction to modify the WU_SRC_[1:5]_LVL bits and a second instruction to modify the WU_SRC_[1:5]_EN bits.

Real Time Clock (RTC)

Offset 0x28Access: User read/write

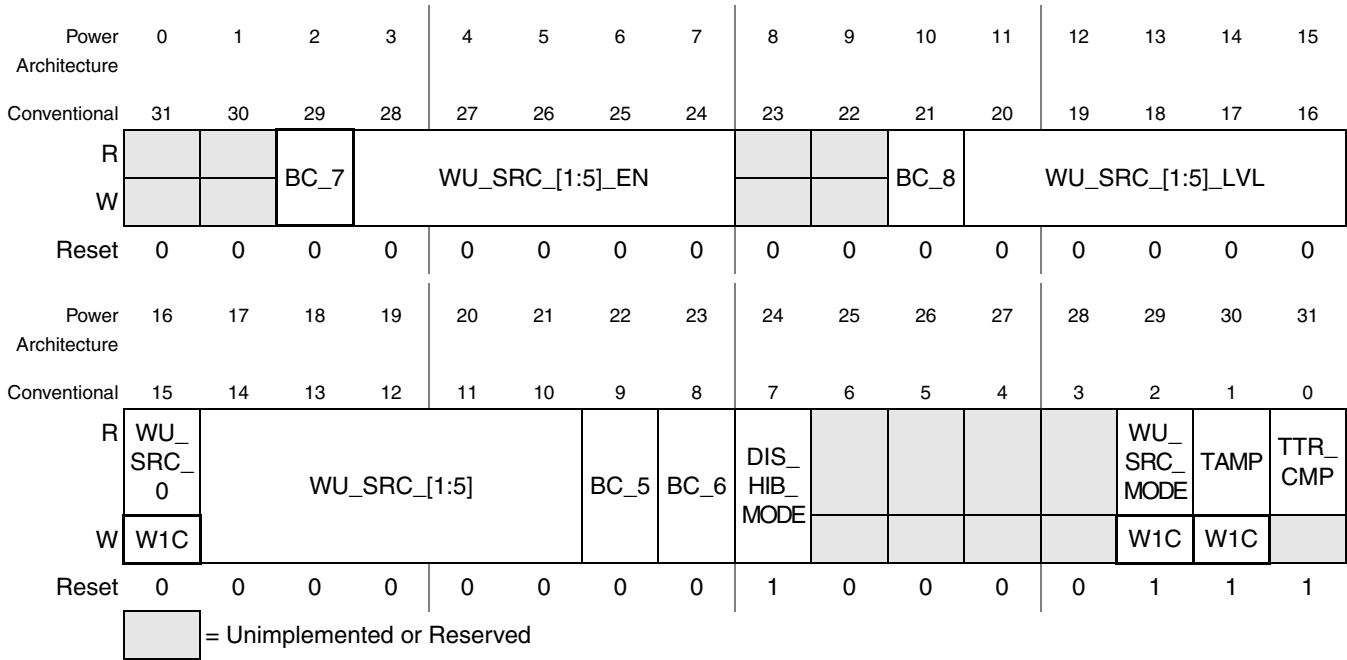


Figure 32-22. RTC Keep Alive Register
(The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 1 of 6)

Field	Description
BC_[8:5]	Breadcrumb bits. This bits are cleared when VBAT_RTC is applied.

Figure 32-23.

Offset 0x28 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W			BC_7													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WU_SRC_0								DIS_HIB_MODE					WU_SRC_MODE	TAMP	TTR_CMP
W	W1C													W1C	W1C	
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

= Unimplemented or Reserved

Figure 32-22. RTC Keep Alive Register
 (The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 2 of 6)

Field	Description
WU_SRC_1_EN	This bit determines the modes of the WU_SRC_1 bit. 1 - WU_SRC_1 functions as a wakeup source (GPI30). 0 - WU_SRC_1 functions as a breadcrumb bit (BC_0).
WU_SRC_2_EN	This bit determines the modes of the WU_SRC_2 bit. 1 - WU_SRC_2 functions as a wakeup source (GPI29). 0 - WU_SRC_2 functions as a breadcrumb bit (BC_1).
WU_SRC_3_EN	This bit determines the modes of the WU_SRC_3 bit. 1 - WU_SRC_3 functions as a wakeup source (GPI28). 0 - WU_SRC_3 functions as a breadcrumb bit (BC_2).
WU_SRC_4_EN	This bit determines the modes of the WU_SRC_4 bit. 1 - WU_SRC_4 functions as a wakeup source (CAN2_RX). 0 - WU_SRC_4 functions as a breadcrumb bit (BC_3).
WU_SRC_5_EN	This bit determines the modes of the WU_SRC_5 bit. 1 - WU_SRC_5 functions as a wakeup source (CAN1_RX). 0 - WU_SRC_5 functions as a breadcrumb bit (BC_4).
WU_SRC_1_LVL	Program the level of the wakeup source, GPI30, as active high or active low. 0 - GPI30 is active low. 1 - GPI30 is active high. Note: This must be configured before WU_SRC_1_EN is asserted to 1.

Figure 32-23.

Real Time Clock (RTC)

Offset 0x28 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R			BC_7	WU_SRC_[1:5]_EN						BC_8	WU_SRC_[1:5]_LVL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WU_SRC_0	WU_SRC_[1:5]						BC_5	BC_6	DIS_HIB_MODE				WU_SRC_MODE	TAMP	TTR_CMP
W	W1C													W1C	W1C	
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

= Unimplemented or Reserved

Figure 32-22. RTC Keep Alive Register
(The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 3 of 6)

Field	Description
WU_SRC_2_LVL	Program the level of the wakeup source, GPI29, as active high or active low. 0 - GPI29 is active low. 1 - GPI29 is active high. Note: This must be configured before WU_SRC_2_EN is asserted to 1.
WU_SRC_3_LVL	Program the level of the wakeup source, GPI28, as active high or active low. 0 - GPI28 is active low. 1 - GPI28 is active high. Note: This must be configured before WU_SRC_1_EN is asserted to 1.
WU_SRC_4_LVL	Program the level of the wakeup source, CAN2_RX, as active high or active low. 0 - CAN2_RX is active low. 1 - CAN2_RX is active high. Note: This must be configured before WU_SRC_1_EN is asserted to 1.
WU_SRC_5_LVL	Program the level of the wakeup source, CAN1_RX, as active high or active low. 0 - CAN1_RX is active low. 1 - CAN1_RX is active high. Note: This must be configured before WU_SRC_1_EN is asserted to 1.
WU_SRC_0	Bit functions as a wakeup source sticky bit. When GPI31 is asserted low then this bit is set to 1. Software can clear it by writing a 1 to this location when GPI31 is negated high.

Figure 32-23.

Offset 0x28Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W			BC_7													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WU_SRC_0								DIS_HIB_MODE					WU_SRC_MODE	TAMP	TTR_CMP
W	W1C													W1C	W1C	
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

= Unimplemented or Reserved

Figure 32-22. RTC Keep Alive Register
(The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 4 of 6)

Field	Description
WU_SRC_1	<p>WU_SRC_1_EN=0: Breadcrumb bit (BC_0) function</p> <p>WU_SRC_1_EN=1: Bit functions as a wakeup source sticky bit. When GPI30 is asserted, according to WU_SRC_1_LVL bit, then this bit is set to 1. Software can clear it by writing a 1 to this location when GPI30 is its inactive wake up state. Note: This bit CANNOT be written at the same time WU_SRC_1_EN is written.</p>
WU_SRC_2	<p>WU_SRC_2_EN=0: Breadcrumb bit (BC_1) function</p> <p>WU_SRC_2_EN=1: Bit functions as a wakeup source sticky bit. When GPI29 is asserted, according to WU_SRC_2_LVL bit, then this bit is set to 1. Software can clear it by writing a 1 to this location when GPI29 is its inactive wake up state. Note: This bit CANNOT be written at the same time WU_SRC_2_EN is written.</p>
WU_SRC_3	<p>WU_SRC_3_EN=0: Breadcrumb bit (BC_2) function</p> <p>WU_SRC_3_EN=1: Bit functions as a wakeup source sticky bit. When GPI28 is asserted, according to WU_SRC_3_LVL bit, then this bit is set to 1. Software can clear it by writing a 1 to this location when GPI28 is its inactive wake up state. Note: This bit CANNOT be written at the same time WU_SRC_3_EN is written.</p>

Figure 32-23.

Real Time Clock (RTC)

Offset 0x28Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R			BC_7	WU_SRC_[1:5]_EN						BC_8	WU_SRC_[1:5]_LVL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WU_SRC_0	WU_SRC_[1:5]						BC_5	BC_6	DIS_HIB_MODE				WU_SRC_MODE	TAMP	TTR_CMP
W	W1C													W1C	W1C	
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1


 = Unimplemented or Reserved

Figure 32-22. RTC Keep Alive Register
(The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 5 of 6)

Field	Description
WU_SRC_4	WU_SRC_4_EN=0: Breadcrumb bit (BC_3) function WU_SRC_4_EN=1: Bit functions as a wakeup source sticky bit. When CAN2_RX is asserted, according to WU_SRC_4_LVL bit, then this bit is set to 1. Software can clear it by writing a 1 to this location when CAN2_RX is its inactive wake up state. Note: This bit CANNOT be written at the same time WU_SRC_4_EN is written.
WU_SRC_5	WU_SRC_5_EN=0: Breadcrumb bit (BC_4) function WU_SRC_5_EN=1: Bit functions as a wakeup source sticky bit. When CAN1_RX is asserted, according to WU_SRC_4_LVL bit, then this bit is set to 1. Software can clear it by writing a 1 to this location when CAN1_RX is its inactive wake up state. Note: This bit CANNOT be written at the same time WU_SRC_4_EN is written.
DIS_HIB_MODE	This bit determines whether to disable the HIB_MODE output pin. 0 - enable the HIB_MODE output pin. 1 - disable the HIB_MODE output pin. The HIB_MODE output pin is always negated high regardless the wake up sources (TTR, GPI[31:28], CAN[1:2]_RX).

Figure 32-23.

Offset 0x28 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W			BC_7													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WU_SRC_0								DIS_HIB_MODE					WU_SRC_MODE	TAMP	TTR_CMP
W	W1C													W1C	W1C	
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1

= Unimplemented or Reserved

Figure 32-22. RTC Keep Alive Register
 (The register is repeated for reference.)

Table 32-13. RTC Keep Alive Register Field Descriptions (Sheet 6 of 6)

Field	Description
WU_SRC_MODE	Wake Up Source Mode bit 0 - allow all proceeding wake up signals to be registered. 1 - inhibit all proceeding wake up signals after the first WU_SRC status bit is registered. Note: In cases two or more wake up signals occur within a 32 kHz window, both signals are registered in their corresponding WU_SRC bits even when WU_SRC_MODE is asserted to 1.
TAMP	This bit is set when VBAT_RTC is applied or oscillator stops running. Software can clear this bit by writing an one to this bit. Software can use this bit recognize loss of VBAT_RTC power or stop of RTC OSC.
TTR_CMP	This bit is the value of the TTR compared to the ATC. TTR_CMP = TTR < ATC

Figure 32-23.

NOTE

- This register has no influence on the Deep Sleep mode, but does have influence on the Hibernation mode.
- GPI31 wake up source cannot be masked nor configured to different polarity level. To enter Hibernation mode, this pin needs to be in a logic one state.

32.4 Functional Description

32.4.1 Behavior at Power On

When power is first applied to the RTC, the target time register is initialized to 0x0000 0000 and the actual time count register is initialized to 0xFFFF FFFE. When the target time register is set to 0x0000 0000, the HIB_MODE pin cannot be asserted. The Keep alive register is initialized to 0x0000 0087. This effectively masks off any external wakeup sources. To activate any of the external wakeup sources associated with the RTC module after power is first applied to VBAT_RTC, software must appropriately program the RTC keep alive register.

After power is applied to the MPC5121e, system software must initialize all of the various registers of the RTC module to properly reflect the current date and time.

32.4.2 Behavior of Wakeup Sources

The RTC employs two basic ways to bring the MPC5121e from a powerdown condition to a fully operational condition. One method is for the value in the actual time count register to increment to a value greater than or equal to the value in the target time register. The second method is to assert one of the external RTC wakeup pins.

From the release of reset, the time target register is set to 0x0000 0000. Thus, the actual time count register is always greater than or equal to the time target register. Under this condition, the HIB_MODE asserts to a logic 1 and can be used to turn on external power regulators powering the device. Recall that the actual time count register is set to 0xFFFF FFFE at the release of reset and cannot be modified by software. When being clocked at a 1 Hz rate, this register requires 136 years before the actual time count rollover. When programming a time in the future to wakeup, the system software reads the actual time count register, adds an appropriate offset and then writes the resultant value to the time target register.

When the value in the actual time count register is less than the value in the target time register, the logic level on the HIB_MODE pin is determined by the WU_SRC_[0:5] bits in the RTC keep alive register. The truth table for controlling the HIB_MODE pin is shown in [Table 32-14](#) and [Table 32-16](#).

Five of the external RTC wakeup pins are controlled by its own enable bit and edge detect bit. If an external RTC wakeup pin is not enabled, then its corresponding WU_SRC_[0:5] bit is held at a logic 0. If an external RTC wakeup pin is enabled, then its corresponding WU_SRC_[0:5] bit is set or cleared according to the truth table presented in [Table 32-16](#).

Table 32-14. Truth Table for the Functionality of the HIB_MODE

Condition	WU_SRC_[0:5]	<u>HIB_MODE</u>
If ATC > TTR	X	1
If ATC = TTR	X	1
If ATC < TTR	0	0
If ATC < TTR	1	1

Table 32-15. Truth Table for the Functionality of the RTC Wakeup Sources

WU_SRC_[1:5]_EN	WU_SRC_[1:5]_LVL	GPI[30:28], CAN[1:2]_RX	WU_SRC_[1:5]
0	X	X	Breadcrumb function
1	0	1 -> 0	1
1	0	0 -> 1	Can be cleared by SW
1	1	0 -> 1	1
1	1	1 -> 0	Can be cleared by SW

Table 32-16. Truth Table for the Functionality of GPI31 Wakeup Sources

GPI31	WU_SRC_0
1 -> 0	1
0 -> 1	Can be cleared by SW

In cases two or more SET_WU_SRC signals occur simultaneously in a 32 kHz window, both signals are registered in their corresponding WU_SRC bits even when WU_SRC_MODE is asserted to 1.

32.4.3 Behavior During Power Off (Hibernation Mode)

The behavior of the RTC is illustrated in [Figure 32-24](#) which shows power being applied to the device, removing power by asserting the $\overline{\text{HIB_MODE}}$ pin and then returning to a power-on condition. Power is applied to $V_{\text{BAT_RTC}}$. This causes the internal RTC_POR signal to the RTC to be released. RTC_POR, shown in [Figure 32-24](#), puts the RTC in its active state when the internal RTC_POR signal is at a logic 0. The actual time count register (ATC) is reset to 0xFFFF FFFE and the time target register (TTR) is reset to 0x0000 0000 at the assertion of the internal reset signal. At this time, the $\overline{\text{HIB_MODE}}$ signal is asserted to a logic 1 indicating that the MPC5121e is not in the hibernate mode. Because the time target register is initialized to 0x0000 0000, the actual time count register is always greater than or equal the time target register as long as the TTR remains at its initialized value of 0x0000 0000. In this case, the $\overline{\text{HIB_MODE}}$ pin remains asserted to a logic 1. Thus, from the assertion of the internal reset signal which is caused by the application of power to $V_{\text{BAT_RTC}}$, the RTC is in an active state and the $\overline{\text{HIB_MODE}}$ pin is asserted to a logic 1 and it remains asserted until the CPU writes a non-zero value to the time target register.

In an actual application, the $\overline{\text{HIB_MODE}}$ pin is used to turn external power regulators on and off the supply power to the MPC5121e.

The assertion of the $\overline{\text{HIB_MODE}}$ pin does not affect the operation of the CPU. For instance, if the $\overline{\text{HIB_MODE}}$ pin is not connected to anything, i.e., an external power regulator, then neither the assertion of the $\overline{\text{HIB_MODE}}$ pin nor the mechanisms that cause the assertion of the $\overline{\text{HIB_MODE}}$ pin have any effect on the operation of MPC5121e.

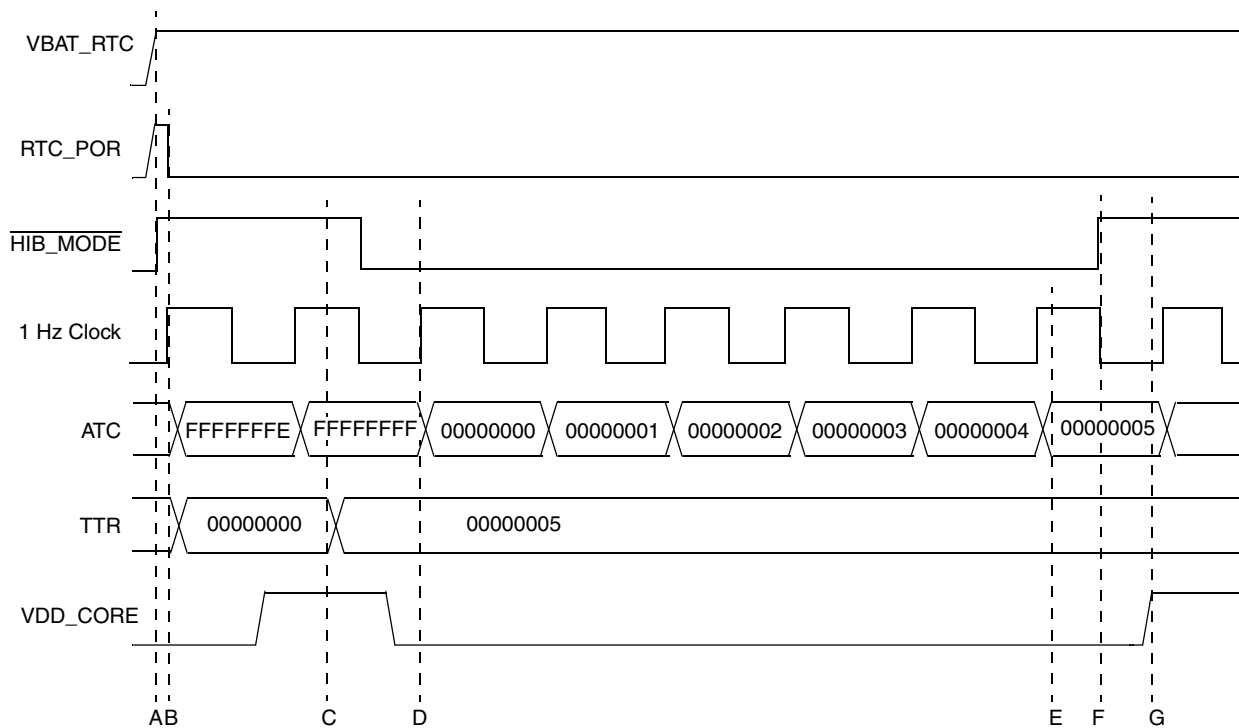


Figure 32-24. Timing Diagram of the RTC in Deep Sleep Mode

- Point A** Power is applied to the VBAT_RTC pin. The application of power on VBAT_RTC caused the internal reset signal to assert to a logic 0 at point A. When the internal reset signal is asserted, the actual time count register is initialized to 0xFFFF FFFE and the target time register is initialized to 0x0000 0000. Until these registers are modified with user software, the target time register is always less than or equal to the actual time count register. Under this condition, the $\overline{\text{HIB_MODE}}$ pin asserts to a logic 1. If the $\overline{\text{HIB_MODE}}$ pin is connected to an external power regulator, the regulator is turned on, thus supplying power to the MPC5121e.
- Point B** The internal RTC_POR signal asserts to a logic 0, which negates the internal reset and enables the RTC. When the internal RTC_POR signal is released, the actual time count register is initialized to 0xFFFF FFFE and the target time register is initialized to 0x0000 0000. Until these registers are modified with user software, the target time register is less than or equal to the actual time count register. Under this condition, the $\overline{\text{HIB_MODE}}$ pin asserts to a logic 1. If the $\overline{\text{HIB_MODE}}$ pin is connected to an external power regulator, the regulator is turned on, thus supplying power to the MPC5121e.
- Point C** The CPU writes 0x0000 0005 to the time target register and enables $\overline{\text{HIB_MODE}}$ function by setting DIS_HIB_MODE bit to zero.
- Point D** The time target register value is greater than the actual time count register. In the diagrams, however, the TTR value is 0x5, whereas the ACT value is 0xFFFFFFFF. This means the ACT is greater than the TTR. This transfer occurs on the next falling edge of the internal 1 Hz clock signal which increments the ATC value. Because the time target register value is greater than the actual time count register value, the $\overline{\text{HIB_MODE}}$ pin is driven to a logic 0. If the $\overline{\text{HIB_MODE}}$ pin is connected to an external power regulator, the regulator is turned off, thus turning off power

to the MPC5121e. VDD_CORE is turned off in response to the $\overline{\text{HIB_MODE}}$ pin being driven to a logic 0.

Point E The actual time count register increments to 0x0000 0005 and now matches the contents of the target time register.

Point F The $\overline{\text{HIB_MODE}}$ pin is driven to a logic 1 on the next negative edge of the 1 Hz

Point G VDD_CORE is turned on in response to the $\overline{\text{HIB_MODE}}$ pin being driven to a logic 1.

NOTE

This scenario assumes the wake up input sources are in an inactive state or disabled by the corresponding WU_SRC_EN bit.

32.4.4 RTC Response to Target Time Register/Actual Time Count Register and External Wakeup Sources

The timing diagram in [Figure 32-25](#) shows the functionality of the target time register (TTR) and the functionality of detecting the positive edge of wakeup source, GPI30. The TTR is originally reset to 0x00000000. When TTR is updated to 0x00000005, then $\text{ATC} < \text{TTR}$ and $\overline{\text{HIB_MODE}}$ is asserted low at the negative edge of the 1 Hz clock.

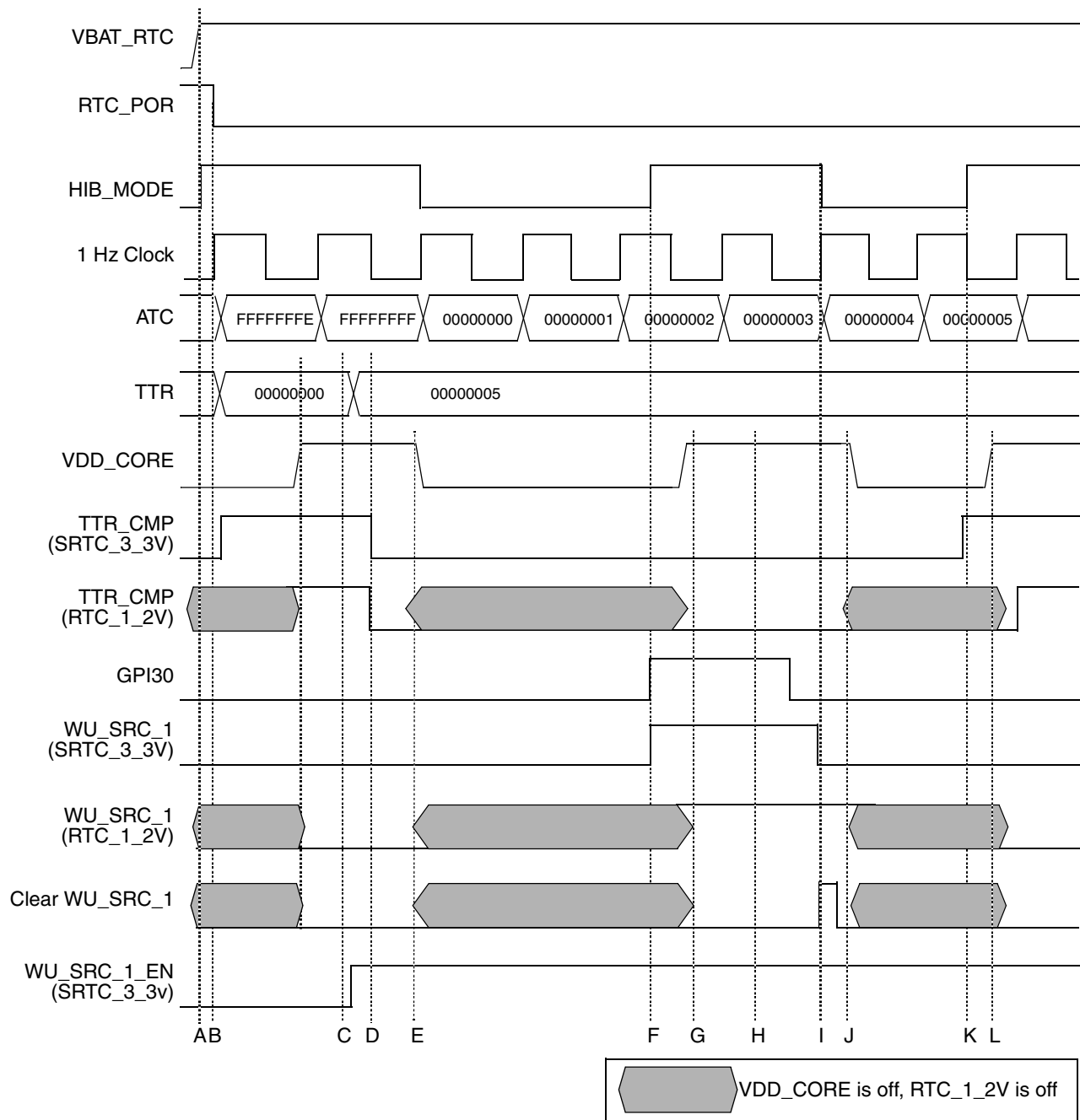


Figure 32-25. Timing Diagram of the Target Time Register (TTR) and the Positive Edge of GPI30

The following is an explanation of [Figure 32-25](#) timing diagram illustrating the functionality of the target time register (TTR) and the functionality of detecting the positive edge of wakeup source, GPI30.

Point A Power is applied to the VBAT_RTC pin. The application of power on VBAT_RTC caused the internal RTC_POR pin to assert to a logic 0 at Point B.

Point B The internal RTC_POR signal asserts to a logic 0 which enables the real time clock. When the internal RTC_POR signal is released, the actual time count register is initialized to 0xFFFF FFFE and the target time register is initialized to 0x0000 0000. Until these registers are modified with

user software. the target time register is less than or equal to the actual time count register. Under this condition, the $\overline{\text{HIB_MODE}}$ pin asserts to a logic 1. If the $\overline{\text{HIB_MODE}}$ pin is connected to an external power regulator, the regulator is turned on, thus supplying power to the MPC5121e.

- Point C The CPU writes 0x0000 0005 to the time target shadow register. Next, the CPU writes to the RTC keep alive register and enables external wakeup source 1. Then, it enables $\overline{\text{HIB_MODE}}$ function by setting DIS_ $\overline{\text{HIB_MODE}}$ bit to zero.
- Point D Now, the time target register is greater than the actual time count register. This transfer occurs on the next falling edge of the internal 1 Hz clock signal which increments the ATC value. Because the time target register value is greater than the actual time count register value, the $\overline{\text{HIB_MODE}}$ pin is driven to a logic 0. If the $\overline{\text{HIB_MODE}}$ pin is connected to an external power regulator, the regulator is turned off, thus turning off power to the MPC5121e.
- Point E VDD_CORE is turned off in response to the $\overline{\text{HIB_MODE}}$ pin being driven to a logic 0.
- Point F The GPI30 is asserted which causes the WU_SRC_1 bit in the RTC Keep Alive Register to assert. This bit remains asserted, regardless of the logic level on the GPI30, until software clears the bit by writing a 1 to the WU_SRC_1 bit position. The assertion of the GPI30 causes the $\overline{\text{HIB_MODE}}$ pin to assert to a logic 1, thus turning on the external power regulators to the MPC5121e.
- Point G The external voltage regulators turn on and apply power to the MPC5121e.
- Point H The CPU writes a 1 to the WU_SRC_1 bit in the RTC keep alive register to negate this bit.
- Point I An internal clear signal clears the WU_SRC_1 bit in the RTC keep alive register. At this time, the WU_SRC_1_EN bit is set, thus enabling GPI30 events to wakeup the device. Also, the actual time count register is less than the target time register. Under these conditions, the $\overline{\text{HIB_MODE}}$ pin asserts to a logic 0, thus turning off the external power regulators to the device.

CAUTION

It is important to keep track of the values in the target time register and the actual time count register when enabling and disabling the external wakeup sources to keep from inadvertently asserting the $\overline{\text{HIB_MODE}}$ pin to a logic 0 and thus turning off the power to the MPC5121e.

- Point J The external voltage regulators turn off and power is removed from the MPC5121e.
- Point K The actual time count register has incremented to 0x0000 0005 and is equal to the target time register. On the next negative edge of the 1 kHz clock after the ATC value is equal to the TTR value, the $\overline{\text{HIB_MODE}}$ pin asserts to a logic 1 causing the external power regulators to turn on.
- Point L The external voltage regulators turn on and apply power to the MPC5121e.

32.4.5 RTC Response to External Wakeup Sources

The following is an explanation of [Figure 32-26](#) timing diagram illustrating the functionality of detecting GPI[30:28] with WU_SRC_MODE disabled.

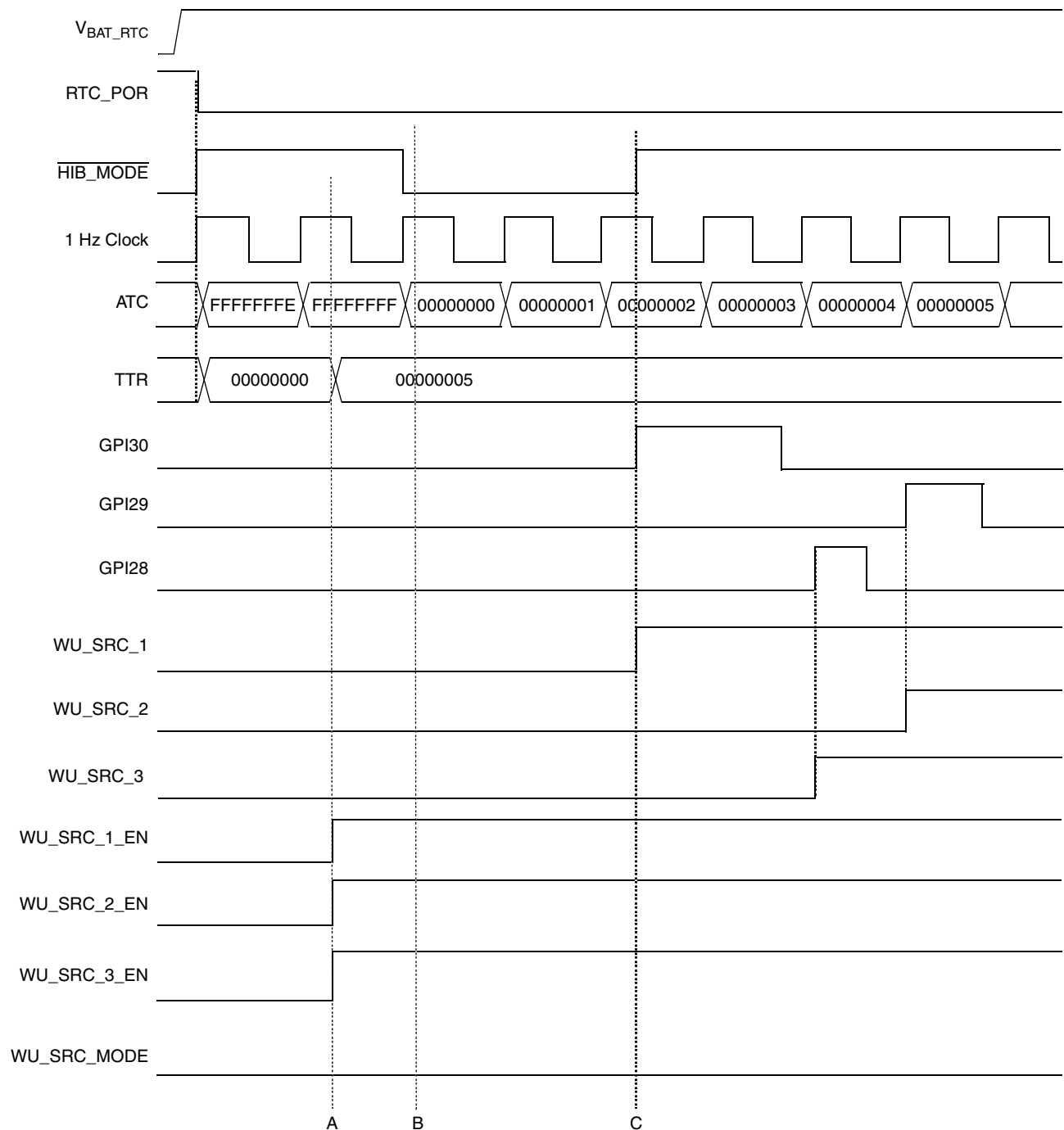


Figure 32-26. Timing Diagram Showing the Functionality of Detecting GPI[30:28] with WU_SRC_MODE Disabled

- Point A The $\overline{\text{HIB_MODE}}$ signal is asserted in response to the release of the internal reset signal. The WU_SRC_1_EN , WU_SRC_2_EN and WU_SRC_3_EN bits are set in the RTC keep alive register to enable the respective external wakeup sources.
- Point B The target time register is written to 0x0000 0005 and is greater than the ATC register. This causes the $\overline{\text{HIB_MODE}}$ pin to assert to a logic 0.
- Point C The GPI30 pin asserts to a logic 1 and causes the $\overline{\text{HIB_MODE}}$ pin to assert to a logic 1. This causes the external power regulators to turn on and apply power to the MPC5121e.

32.4.5.1 Behavior of the RTC keep alive register WU_SRC_x bits when the WU_SRC_MODE is disabled

As shown in [Figure 32-26](#), the WU_SRC_MODE is disabled. The $\overline{\text{HIB_MODE}}$ pin is asserted to a logic 0 as soon as software writes the target time register to 0x0000 0005. The external RTC wakeup source pin, GPI30 pin asserts. As soon as this pin asserts, the corresponding bit in the RTC keep alive register is set. Likewise, when GPI29 and GPI28 are set, their corresponding bits, WU_SRC_2 and WU_SRC_3 are also set. If the system software accesses the RTC Keep Alive Register and more than one of the wakeup bits is set, it is not possible to see which one occurred first. On the other hand, any time that one of the six external RTC wakeup pins assert to an active state, that occurrence is recorded in the RTC keep alive register. Recall that the WU_SRC_x bits are sticky bits and does not return to a logic 0 until a logic 1 is written to their respective bit position in the RTC keep alive register.

32.4.5.2 Behavior of the RTC keep alive register WU_SRC_x bits when the WU_SRC_MODE is enabled

As shown in [Figure 32-27](#), the WU_SRC_MODE is enabled. The $\overline{\text{HIB_MODE}}$ pin is asserted to a logic 0 as soon as software writes the target time register to 0x0000 0005. The external RTC wakeup source pin, GPI30 pin asserts. As soon as this pin asserts, the corresponding bit in the RTC Keep Alive Register is set; however, when GPI29 and GPI28 are set after GPI30 is set and remains set, the WU_SRC_2 and WU_SRC_3 are not set. In this case, once an external RTC wakeup source is asserted and the corresponding bit in the RTC Keep Alive Register is set, no more bits is set in the RTC keep alive register by asserting additional external RTC wakeup pins until the first external interrupt source is negated and its corresponding flag in the RTC Keep Alive Register is also negated by writing a logic 1 to its bit position. This methodology allows the system software to determine which external wakeup source caused the device to exit the hibernate mode. After the bit in the RTC Keep Alive Register is cleared that caused the device to exit the hibernate mode, then next external wakeup source to assert is recognized and then all further wakeup sources is inhibited until the current wakeup source is negated and its associated status bit is cleared. Recall that the WU_SRC_x bits are sticky bits and does not return to a logic 0 until a logic 1 is written to their respective bit position in the RTC Keep Alive Register.

Real Time Clock (RTC)

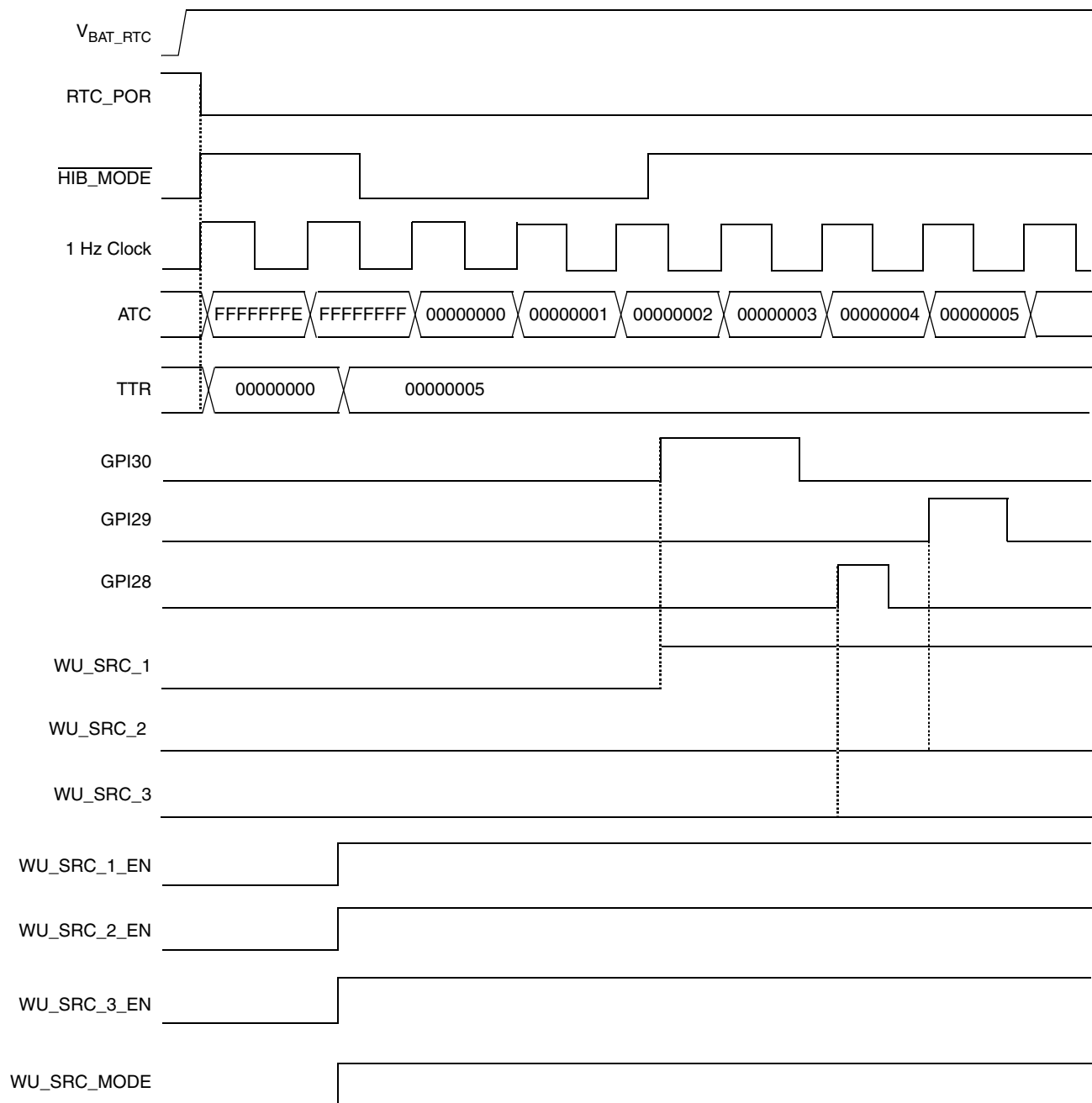


Figure 32-27. Timing Diagram Illustrating the Functionality of Detecting GPI[30:28] with WU_SRC_MODE Enabled

Chapter 33

SATA Controller (SATA)

33.1 Introduction

The SATA I is a high-speed serialized ATA data link interface compliant with SATA Revision 1.0a by SerialATA workgroup. It includes DMA controller, command layer, transport layer, link layer, PhyCtrl layer, PCS(Physical Coding Sublayer) and physical layer. The SATA controller supports one SATA device only.

33.1.1 Features

- Compliant with SATA 1.0a spec
 - Supports:
 - Register FIS
 - Set Device Bits FIS
 - PIO Setup FIS
 - DMA Activate FIS
 - BIST Activate FIS
 - DATA FIS
 - First-party DMA access (DMA setup FIS) is not supported.
- Embedded dedicated DMA for data transfer
- Support device retimed loopback and device transmit only modes.
- Single-channel SATA-I PHY
- Power efficient normal operation mode of SATA PHY
- Partial and Slumber power management modes of SATA PHY
- Receiver of SATA PHY capable of tracking spread spectrum clocking

33.1.2 Modes of Operation

This SATA supports three operation modes: function mode, BIST mode and power management mode.

In function mode, the SATA host transmits and receives data to or from the attached mass storage device. Refer to [Section 33.4, “Functional Description,”](#) for detailed functionality description.

In BIST mode, the transmitter and receiver of host are tested by internal far-end loopback mode or near-end loopback mode. The power mode of SATA block includes:

1. Normal: full-duplex serial data transmitted at 1.5GHz.
2. Slumber: power management state with slower wakeup time.
3. Partial: power management state with faster wakeup time.

33.2 External Signal Description

Table 33-2. External Signal Properties

Name	Description	I/O	Reset
SATA_XTALI	Clock input to PLL in SATA PHY	I	—
SATA_XTALO	Clock output of PLL in SATA PHY	O	—
SATA_RESREF	External resistor reference input.	I	—
SATA_TXP	Transmitter positive differential pair output.	O	Idle State
SATA_TXN	Transmitter minus differential pair output.	O	Idle State
SATA_RXP	Receiver positive differential pair input.	I	Off
SATA_RXN	Receiver minus differential pair input.	I	Off
SATA_ANAVIZ	Analog test output, mux of several analog test points within the PHY. Selected with anaDebugIn[28:25]: 0: Tri-state anaViz output 1: Bandgap voltage (vbg) 2: 50uA resRef resistor current (i50u_rpoly) 3: 8uA bandgap current (i8u_vtemp) 4: Regulated PLL voltage (vreg_pll) 5: Analog 1.2V supply (VDDA_1P2) 6: Voltage regulator supply (VDDA_REG) 7: Analog 3.3V supply (VDDA_3P3) An extra bit anaDebugIn[13] is used to increase the visibility on the SATA_ANAVIZ.	O	Bandgap Voltage

33.3 Memory Map and Register Definition

Table below shows memory map for SATA host registers.

NOTE

All registers use little-endian ordering. Software running on the local processor in big-endian mode must byte-swap the data.

After reset, only additional registers can be accessed. All registers can be accessed when bit `sata_ctl_en` in PMCtrl register is set to 1.

In this documentation, a doublewords means 4 bytes.

Table 33-3. Block Memory Map

Offset or Address	Register	Access	Section/Page
ATA Taskfile and Control Registers			
0x0000_0000	DATA. Data port register	R/W	33.3.1.1/33-10
0x0000_0004	Error_Features. Features/Error register	R/W	33.3.1.1/33-10
0x0000_0008	Sector_Count. Sector count register	R/W	33.3.1.1/33-10
0x0000_000c	Sector_Number. Sector number register	R/W	33.3.1.1/33-10
0x0000_0010	Cylinder_Low. Cylinder low register	R/W	33.3.1.1/33-10
0x0000_0014	Cylinder_High. Cylinder high register	R/W	33.3.1.1/33-10
0x0000_0018	Device_Head. Device/Head register	R/W	33.3.1.1/33-10
0x0000_001c	Status_Command. Status/Command register	R/W	33.3.1.1/33-10
0x0000_0048	Altstat_Devctrl. Alternate status /device control	R/W	33.3.1.1/33-10
SATA Status and Control Registers			
0x0000_0100	SStatus. SATA status register	R	33.3.1.2.1/33-11
0x0000_0104	SError. SATA error register	R/W	33.3.1.2.2/33-13
0x0000_0108	SControl. SATA control register	R/W	33.3.1.2.3/33-16
Special Registers			
0x0000_0140	Trans_cfg. Transport layer config register	R/W	33.3.1.3.1/33-17
0x0000_0144	Trans_status0. Transport layer status register 0	R	33.3.1.3.2/33-18
0x0000_0148	Link_cfg0. Link layer config register 0	R/W	33.3.1.3.3/33-20
0x0000_014c	Link_cfg1. Link layer config register 1	R/W	33.3.1.3.4/33-24
0x0000_0150	Link_cfg2. Link layer config register 2	R/W	33.3.1.3.5/33-25
0x0000_0154	Link_status0. Link layer status register 0	R	33.3.1.3.6/33-26
0x0000_0158	Link_status1. Link layer status register 1	R	33.3.1.3.7/33-27
0x0000_0174	Phy_cfg. Phy layer config register	R/W	33.3.1.3.8/33-28

Table 33-3. Block Memory Map (continued)

Offset or Address	Register	Access	Section/Page
0x0000_0178	Trans_status1. Transport layer status register 1	R	33.3.1.3.9/33-29
0x0000_017c	Tx_bist_dat0. Bist data 0 register	R/W	33.3.1.3.10/33-30
0x0000_0180	Bist_dat0_sel. Bist data 0 select register	R/W	33.3.1.3.11/33-30
0x0000_0184	Tx_bist_dat1. Bist data 1 register	R/W	33.3.1.3.12/33-31
0x0000_0188	Bist_dat1_sel. Bist data 1 select register	R/W	33.3.1.3.13/33-31
0x0000_018c	Tx_bist_mode. Bist mode register	R/W	33.3.1.3.14/33-32
0x0000_0190	Bist_written. Bist written register	R/W	33.3.1.3.15/33-33
0x0000_0194	Bist_status. Bist status register	R	33.3.1.3.16/33-33
0x0000_0198	Tx_send_dmat. Send DMAT register	R/W	33.3.1.3.17/33-34
0x0000_019c	Atapi_en. ATAPI enable register	R/W	33.3.1.3.18/33-34
Additional Registers			
0x0000_01c0	rxCdrCtrl. Receiver CDR control register	R/W	33.3/33-3
0x0000_01c4	rxCdrStat. Receiver CDR status register	R	33.3/33-3
0x0000_01c8	rxCtrl. Receiver register	R/W	33.3/33-3
0x0000_01cc	txCtrl. Transmitter register	R/W	33.3/33-3
0x0000_01d0	vregLvlCtrl. Voltage regulator level register	R/W	33.3/33-3
0x0000_01d4	PMCtrl. Powermanagement control register	R/W	33.3.1.4.1/33-35
0x0000_01d8	PMStat. Powermanagement status register	R	33.3.1.4.2/33-36
0x0000_01e0	DmaEndian. DMA endian control register	R/W	33.3.1.5.1/33-37
DMA Registers			
0x0000_1080	Dma_stat_ctrl. DMA Status/Control register	R/W	33.3.1.5.2/33-38
0x0000_1084	Dma_remain_ahb. DMA remaining AHB register	R	33.3/33-3
0x0000_1088	Dma_max_cnt. DMA maximum count register	R/W	33.3.1.5.3/33-39
0x0000_108c	Dma_start. DMA start address register	R/W	33.3.1.5.4/33-39
0x0000_1090	Dma_current. DMA current address register	R	33.3/33-3
0x0000_1094	Dma_remain_tc. DMA remaining count register	R	33.3/33-3

Table 33-4 shows SATA the register summary table.

Table 33-4. SATA Register Summary (Sheet 1 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X0000_0000 DATA	R																
	W																
	R	DATA															
	W																
0X0000_0004 ERROR_ FEATURES	R																
	W																
	R														ABRT		
	W									FEATURES							
0X0000_0008 SECTOR_CO UNT	R																
	W																
	R									SEC_CNT							
	W																
0X0000_000C SECTOR_ NUMBER	R																
	W																
	R									SEC_NUM							
	W																
0X0000_0010 CYLINDER_L OW	R																
	W																
	R									CYL_LOW							
	W																
0X0000_0014 CYLINDER_HI GH	R																
	W																
	R									CYL_HIGH							
	W																
0X0000_0018 DEVICE_HEA D	R																
	W																
	R																
	W												DEV				

Table 33-4. SATA Register Summary (Sheet 2 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X0000_001C STATUS_ COMMAND	R																
	W																
	R									BSY	DRDY			DRQ			ERR
	W									COMMAND							
0X0000_0048 ALTSTAT_DE VCTRL	R																
	W																
	R									BSY	DRDY			DRQ			ERR
	W														SRST	NIEN	0
0X0000_0300 SSTATUS	R																
	W																
	R					IPM				SPD				DET			
	W																
0X0000_0104 ERROR	R							F	T	S	H	C	D	B	W	I	N
	W																
	R					E	P	C	T							M	I
	W																
0X0000_0108 SCONTROL	R																
	W																
	R					IPM				SPD				DET			
	W																
0X0000_0140 TRANS_CFG	R	TRANS_CFG															
	W	TRANS_CFG															
	R	TRANS_CFG															
	W	TRANS_CFG															
0X0000_0144 TRANS_STAT US0	R	TRANS_STATUS0															
	W																
	R	TRANS_STATUS0															
	W																
0X0000_0148 LINK_CFG0	R	LINK_CFG0															
	W	LINK_CFG0															
	R	LINK_CFG0															
	W	LINK_CFG0															

Table 33-4. SATA Register Summary (Sheet 3 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X0000_014C LINK_CFG1	R	LINK_CFG1															
	W	LINK_CFG1															
	R	LINK_CFG1															
	W	LINK_CFG1															
0X0000_0150 LINK_CFG2	R	LINK_CFG2															
	W	LINK_CFG2															
	R	LINK_CFG2															
	W	LINK_CFG2															
0X0000_0154 LINK_STATUS 0	R	LINK_STATUS0															
	W																
	R	LINK_STATUS0															
	W																
0X0000_0158 LINK_STATUS 1	R	LINK_STATUS1															
	W																
	R	LINK_STATUS1															
	W																
0X0000_0174 PHY_CFG1	R	PHY_CFG															
	W	PHY_CFG															
	R	PHY_CFG															
	W	PHY_CFG															
0X0000_0178 TRANS_STAT US1	R	TRANS_STATUS1															
	W																
	R	TRANS_STATUS1															
	W																
0X0000_017C TX_BIST_DAT 0	R	TX_BIST_DAT0															
	W	TX_BIST_DAT0															
	R	TX_BIST_DAT0															
	W	TX_BIST_DAT0															
0X0000_0180 BIST_DAT0_S EL	R																
	W																
	R																S
	W																

Table 33-4. SATA Register Summary (Sheet 4 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X0000_0184 TX_BIST_DAT1	R	TX_BIST_DAT1															
	W																
	R	TX_BIST_DAT1															
	W																
0X0000_0188 BIST_DAT1_SEL	R																S
	W																
	R																
	W																
0X0000_018C TX_BIST_MODE	R																TX_BIST_MODE
	W																
	R																
	W																
0X0000_0190 BIST_WRITE_N	R																BW
	W																
	R																
	W																
0X0000_0194 BIST_STATUS	R																BS
	W																
	R																
	W																
0X0000_0198 TX_SEND_DMAT	R																SD
	W																
	R																
	W																
0X0000_019C ATAPI_EN	R																AE
	W																
	R																
	W																
0X0000_01C0 RXCDRCTRL	R	RXCDRCTRL															
	W																
	R	RXCDRCTRL															
	W																

Table 33-4. SATA Register Summary (Sheet 5 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X0000_01C4 RXCDRSTAT	R	RXCDRSTAT															
	W																
	R	RXCDRSTAT															
	W																
0X0000_01C8 RXCTRL	R																
	W																
	R											RXGAINCTRL			RXSQTHCTRL		
	W																
0X0000_01CC TXCTRL	R																
	W																
	R					TXPREEMCTRL						TXAMPCTRL					
	W																
0X0000_01D0 VREGLVLC TRL	R																
	W																
	R														VREGLVLCCTRL		
	W																
0X0000_01D4 PMCTRL	R																
	W																
	R									LBEN ASEL	LBENA_FSL		SATA _CTL_ EN	SATA _PCS _EN	SIGN ALDE TSEL		
	W																
0X0000_01D8 PMSTAT	R																
	W																
	R														RXLO CKON CE DET	TXLO CKON CE DET	
	W																
0X0000_01E0 DMAENDIAN	R																
	W																
	R																DMA
	W																END IAN

Table 33-4. SATA Register Summary (Sheet 6 of 6)

Register Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0X0000_1080 DMA_STAT_C TRL	R		RDPNTRA				RDPNTRB			WRPNTRA			WRPNTRB			BUFSTAT		STALL INT	
	W																		
	R														STALL INTEN	DISK READ	DMA EN		
	W																		
0X0000_1084 DMA_REMAI N_ AHB	R								DMA_REMAIN_AHB										
	W																		
	R	DMA_REMAIN_AHB																	
	W																		
0X0000_1088 DMA_MAX_C NT	R								DMA_MAX_CNT										
	W								DMA_MAX_CNT										
	R	DMA_MAX_CNT																	
	W	DMA_MAX_CNT																	
0X0000_108C DMA_START	R	DMA_START																	
	W	DMA_START																	
	R	DMA_START																	
	W	DMA_START																	
0X0000_1090 DMA_MAX_C NT	R	DMA_CURRENT																	
	W																		
	R	DMA_CURRENT																	
	W																		
0X0000_1094 DMA_REMAI N_TC	R								DMA_REMAIN_TC										
	W																		
	R	DMA_REMAIN_TC																	
	W																		

33.3.1 Register Descriptions

33.3.1.1 ATA Taskfile and Control Registers

The ATA taskfile registers (address offset 0x0000_0000 to 0x0000_001c) send commands to the device or post status from the device. The control registers (address offset 0x0000_0048) are for device control and to post alternate status. No description of these registers is given here. Please consult the ATA/ATAPI specification for the detailed information.

33.3.1.2 SATA Status and Control Registers

SATA host adapters include an additional block of registers mapped separately and independently from the ATA taskfile registers for reporting additional error and status information and to allow control of capabilities unique to SATA. These registers are SATA status and control registers. Sixteen contiguous registers allocated with the first three are defined and the remaining 13 are reserved for future definition. The three defined registers are SStatus, SError, and SControl registers.

Consult section 10.1 in SATA specification (Revision 1.0a) for more details.

33.3.1.2.1 SStatus Register

This 32-bit read-only register conveys the current state of the interface and host adapter. Writes to the register have no effect.

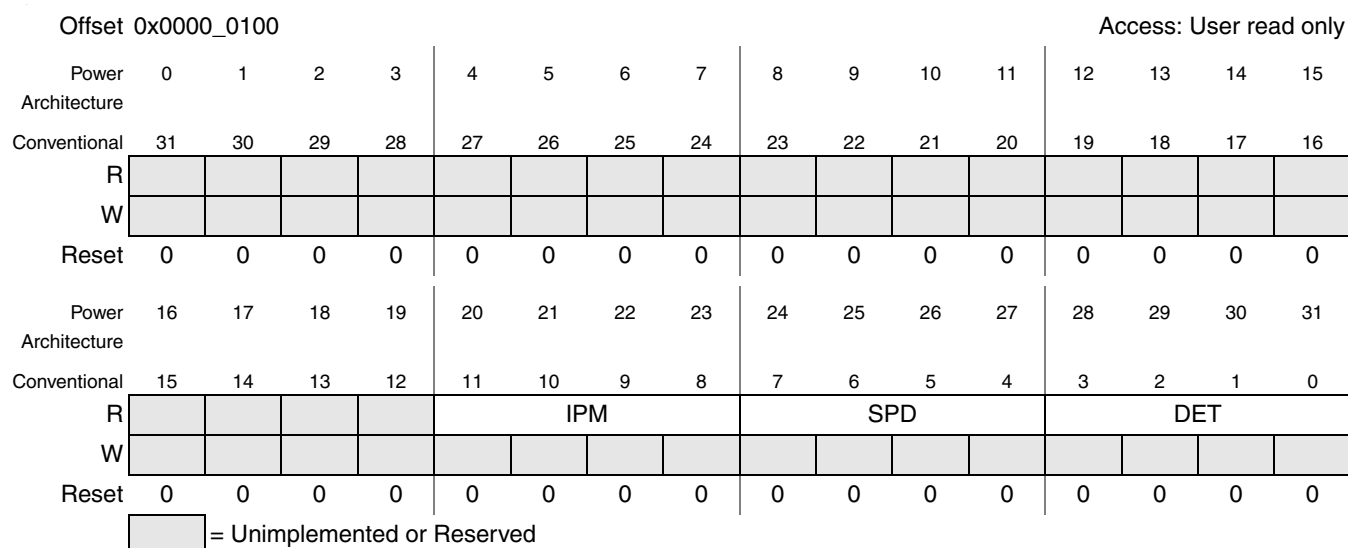


Figure 33-1. SStatus Register

Table 33-5. SStatus Field Descriptions

Field	Description
11-8 IPM	IPM value indicates the current interface power management state. 0000 Device not present or communication not established. 0001 Interface in active state. 0010 Interface in PARTIAL power management state. 0110 Interface in SLUMBER power management state. All other values reserved

Table 33-5. SStatus Field Descriptions (continued)

Field	Description
7-4 SPD	0000 No negotiated speed (device not present or communication not established). 0001 Generation 1 communication rate negotiated SPD value indicates the negotiated interface communication speed established. All other values reserved.
3-0 DET	The DET value indicates the interface device detection and Phy state. 0000) No device detected and Phy communication not established. 0001) Device presence detected but Phy communication not established. 0011) Device presence detected and Phy communication established. 0100) Phy in offline mode as a result of the interface being disabled or running in a BIST loopback mode. All other values reserved

33.3.1.2.2 SError Register

This 32-bit register conveys supplemental interface error information to complement the error information available in the taskfile error register. The register represents all the detected errors accumulated since the last time the SError register was cleared. Set bits in the error register are explicitly cleared by a write operation to the SError register or a reset operation. The value written to clear set error bits shall have ones encoded in the bit positions corresponding to the bits to be cleared. Host software should clear the interface SError register at appropriate checkpoints to best isolate error conditions and the commands they impact.

Offset 0x0000_0104

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							F	T	S	H	C	D	B	W	I	N
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					E	P	C	T							M	I
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 33-2. SError Register
(Register is repeated for reference.)

Table 33-6. SError Field Descriptions (Sheet 1 of 3)

Field	Description
F	Unrecognized FIS type. When set to one, this bit indicates that since the bit was last cleared one or more FISs were received by the transport layer with good CRC, but had a type field that was not recognized.
T	Transport State Transition Error. When set to one, this bit indicates an error has occurred in the transition from one state to another within the transport layer since the last time this bit was cleared.
S	Link Sequence Error. When set to one, this bit indicates that one or more link state machine error conditions was encountered since the last time this bit was cleared. The link layer state machine defines the conditions under which the link layer detects an erroneous transition.
H	Handshake Error. When set to one, this bit indicates that one or more R_ERR handshake response was received in response to frame transmission. Such errors may be the result of a CRC error detected by the recipient, a disparity or 10b/8b decoding error, or other error condition leading to a negative handshake on a transmitted frame.
C	CRC Error. When set to one, this bit indicates that one or more CRC errors occurred with the Link Layer since the bit was last cleared.

Offset 0x0000_0104

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							F	T	S	H	C	D	B	W	I	N
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					E	P	C	T							M	I
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 33-2. SError Register
 (Register is repeated for reference.)

Table 33-6. SError Field Descriptions (Sheet 2 of 3)

Field	Description
D	Disparity Error. When set to one, this bit indicates an incorrect disparity was detected one or more times since the last time the bit was cleared.
B	10b to 8b Decode Error. When set to a one, this bit indicates one or more 10b to 8b decoding errors occurred since the bit was last cleared.
W	Comm Wake. When set to one this bit indicates a comm wake signal was detected by the Phy since the last time this bit was cleared.
I	Phy Internal Error. When set to one, this bit indicates the Phy detected an internal error since the last time this bit was cleared.
N	PhyRdy Change. When set to one, this bit indicates the PhyRdy signal changed state since the last time this bit was cleared.
E	Internal Error. The host bus adapter experienced an internal error that caused the operation to fail and may have put the host bus adapter into an error state. Host software should reset the interface before re-trying the operation. If the condition persists, the host bus adapter may suffer from a design issue rendering it incompatible with the attached device.
P	Protocol Error. A violation of the SATA protocol was detected. This can arise from invalid or poorly formed FISs being received, from invalid state transitions or from other causes. Host software should reset the interface and retry the corresponding operation. If such an error persists, the attached device may have a design issue rendering it incompatible with the host bus adapter.
C	Non-Recovered Persistent Communication or Data Integrity Error. A communication error that was not recovered occurred that is expected to be persistent. Because the error condition is expected to be persistent, the operation does not need to be retried by host software. Persistent communications errors may arise from faulty interconnect with the device, from a device that has been removed or has failed, or a number of other causes.

Offset 0x0000_0104

Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R							F	T	S	H	C	D	B	W	I	N
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					E	P	C	T							M	I
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 33-2. SError Register
(Register is repeated for reference.)

Table 33-6. SError Field Descriptions (Sheet 3 of 3)

Field	Description
T	Non-Recovered Transient Data Integrity Error. A data integrity error occurred that was not recovered by the interface. Because the error condition is not expected to be persistent, the operation should be retried by host software.
M	Recovered Communications Error. Communications between the device and host was temporarily lost, but was re-established. This can arise from a device temporarily being removed, from a temporary loss of Phy synchronization, or from other causes and may be derived from the PhyNRdy signal between the Phy and link layers. No action is required by the host software because the operation ultimately succeeded. However, host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed.
I	Recovered Data Integrity Error. A data integrity error occurred that was recovered by the interface through a retry operation or other recovery action. This can arise from a noise burst in the transmission, a voltage supply variation, or from other causes. No action is required by host software because the operation ultimately succeeded. However, host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed.

33.3.1.2.3 SControl Register

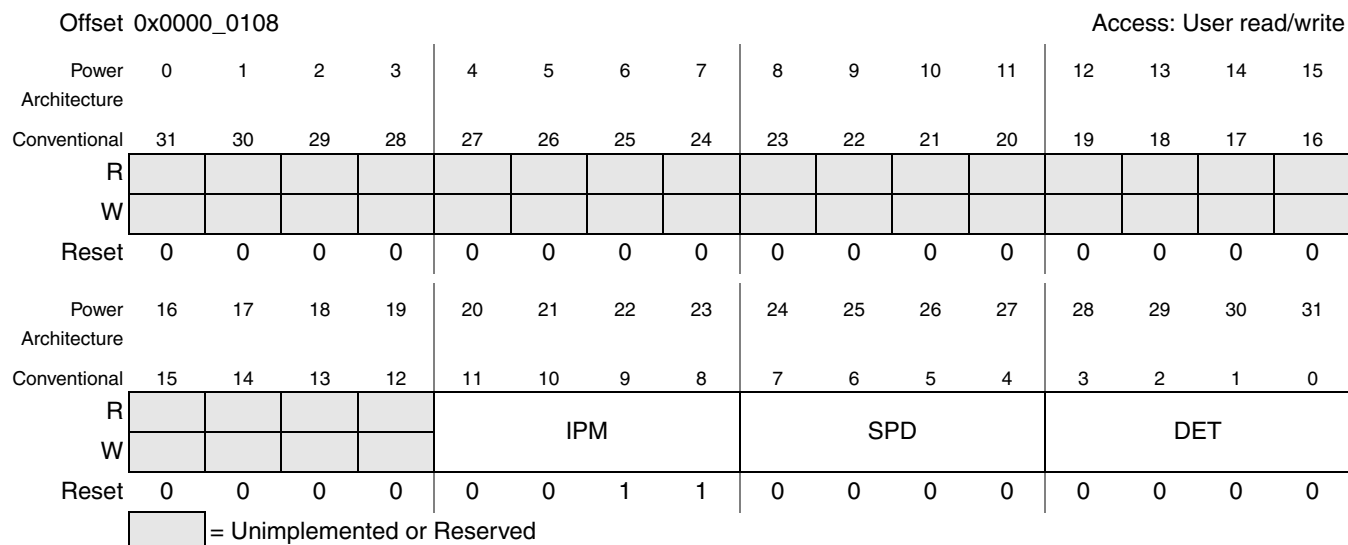


Figure 33-3. SControl Register

Table 33-7. SControl Field Descriptions

Field	Description
11-8 IPM	<p>The IPM field represents the enabled interface power management states that can be invoked via the SATA interface power management capabilities.</p> <p>0000 No interface power management state restrictions.</p> <p>0001 Transitions to the PARTIAL power management state disabled.</p> <p>0010 Transitions to the SLUMBER power management state disabled.</p> <p>0011 Transitions to both the PARTIAL and SLUMBER power management states disabled.</p> <p>All other values reserved.</p>
7-4 SPD	<p>The SPD field represents the highest allowed communication speed the interface is allowed to negotiate when interface communication speed is established.</p> <p>0000 No speed negotiation restrictions.</p> <p>0001 Limit speed negotiation to a rate not greater than Generation 1 communication rate.</p> <p>All other values reserved.</p>
3-0 DET	<p>The DET field controls the host adapter device detection and interface initialization.</p> <p>0000 No device detection or initialization action requested.</p> <p>0001 Perform interface communication initialization sequence to establish communication. This is functionally equivalent to a hard reset and results in the interface being reset and communications reinitialized. Upon a write to the SControl register that sets the LSB of the DET field to one, the host interface shall transition to the HP1: HR_Reset state and shall remain in that state until the LSB of the DET field is cleared to zero by a subsequent write to the SControl register.</p> <p>0100 Disable the SATA interface and put Phy in offline mode.</p> <p>All other values reserved</p>

33.3.1.3 Special Registers

There are some miscellaneous registers that are not part of the standard and are present as a result of the SATA implementation.

33.3.1.3.1 Trans_cfg Register

Offset 0x0000_0140												Access: User read/write					
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R																	
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																	
W							TX_WATERMARK			RX_WATERMARK						BE	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	
	= Unimplemented or Reserved																

Figure 33-4. Trans_cfg Register

Table 33-8. Trans_cfg Field Descriptions

Field	Description
TX_WATERMARK	This defines the watermark of the eight deep Tx FIFO. Because there are internal clock boundaries to be considered, it is recommended that a value of eight not to be used. The initial recommended value is four.
RX_WATERMARK	This sets the number of locations in the 48 deep RX FIFO that can be used before the SATA controller transmits holds to the transmitting device. It takes some time for the holds to get to the other end and that in the interim there must be enough room in the FIFO to absorb all data that could arrive. The maximum value of this depends on the value of the CFG_RXREFMODE bit in Phy_cfg register. CFG_RXREFMODE = 0, Maximum value = 12 CFG_RXREFMODE = 1, Maximum value = 10
BE	Configuration bit to enable entry to BIST mode when BIST Activate FIS is receive.

33.3.1.3.2 Trans_status0 Register

Offset 0x0000_0144

Access: User read only

Power	0	1	2	3	4	5	6	7
Architecture								
conventional	31	30	29	28	27	26	25	24
R						SRST_CLEARED	SRST_SET	TRANSMIT_CRF
W								
Reset	—	—	—	—	—	—	—	—
Power	8	9	10	11	12	13	14	15
Architecture								
conventional	23	22	21	20	19	18	17	16
R	SEND_FIS_SM				GET_FIS_SM			RX_FIFO_EMPTY
W								
Reset	—	—	—	—	—	—	—	—
Power	16	17	18	19	20	21	22	23
Architecture								
conventional	15	14	13	12	11	10	9	8
R	TRANS_READY	R_STATUS						
W								
Reset	—	—	—	—	—	—	—	—
Power	24	25	26	27	28	29	30	31
Architecture								
conventional	7	6	5	4	3	2	1	0
R	R_STATUS	LAST_FIS_EQ_PSUW	LAST_FIS_EQ_PSUR	TRANS_STATE				
W								
Reset	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

Figure 33-5. Trans_status0 Register

Table 33-9. Trans_status0 Field Descriptions

Field	Description
SRST_CLEARED	SRST bit in the device control register in the ATA taskfile register has been cleared.
SRST_SET	SRST bit in the device control register in the ATA taskfile register has been set.
TRANSMIT_CRF	A Control Frame is to be transmitted.
SEND_FIS_SM	Send FIS state machine.
GET_FIS_SM	Get FIS state machine.
RX_FIFO_EMPTY	The receive FIFO is empty.
TRANS_READY	Transport Layer is ready.
R_STATUS	Copy of the the ATA taskfile status register.
LAST_FIS_EQ_PSUW	Last FIS was PIO SU write.

Table 33-9. Trans_status0 Field Descriptions (continued)

Field	Description
LAST_FIS_EQ_PSUR	Last FIS was PIO SU read.
TRANS_STATE	<p>Transport Layer main state machine.</p> <p>5'd0: Not_Ready5'd17: HTDA5</p> <p>5'd1: HTI15'd18: HTPS1</p> <p>5'd2: HTI25'd19: HTPS1A</p> <p>5'd3: HTCM1 5'd20: HTPS2</p> <p>5'd4: HTCM2 5'd21: HTPS3</p> <p>5'd5: HTCR1 5'd22: HTPS4</p> <p>5'd6: HTCR2 5'd23: HTPS4A</p> <p>5'd7: HTXBIST1 5'd24: HTPS5</p> <p>5'd8: HTXBIST2 5'd25: HTPS6</p> <p>5'd9: HTR1 5'd26: HTRBIST1</p> <p>5'd10: HTR2 5'd27: HTRBIST2</p> <p>5'd11: HTDB0 5'd28: Clear_All</p> <p>5'd12: HTDB1 5'd29: HTDR1</p> <p>5'd13: HTDA1 5'd30: HTDR2</p> <p>5'd14: HTDA2</p> <p>5'd15: HTDA3</p> <p>5'd16: HTDA4</p> <p>Consult section 8.6 in SATA specification (Revision 1.0a) for more details about transport layer state machine.</p>

33.3.1.3.3 Link_cfg0 Register

Offset 0x0000_0148

Access: User read/write

Power	0	1	2	3	4	5	6	7
Architecture								
conventionl	31	30	29	28	27	26	25	24
R		CFG_EN_PARTIAL	CFG_LINK_WAKEUP	CFG_LINK_GO_SLUMBER	CFG_LINK_GO_PARTIAL	CFG_PRIM_OVR_EN	CFG_PHY_READY_TIMER	
W								
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventionl	23	22	21	20	19	18	17	16
R	CFG_PHY_READY_TIMER							
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventionl	15	14	13	12	11	10	9	8
R	CFG_ALIGN_RATE							
W								
Reset	1	1	1	1	1	1	1	1
Power	24	25	26	27	28	29	30	31
Architecture								
conventionl	7	6	5	4	3	2	1	0
R	CFG_EN_PHY_TO	CFG_SEND_4_ALIGNS	CFG_EN_SLUMBER	CFG_RX_SCRAMBLE	CFG_TX_SCRAMBLE	CFG_TXPRI_M_JUNK	CFG_TX_CONT	CFG_TX_BADCRC
Reset	0	0	0	1	1	0	1	0
	= Unimplemented or Reserved							

Figure 33-6. Link_cfg0 Register
(Register is repeated for reference.)

Table 33-10. Link_cfg0 Field Descriptions (Sheet 1 of 4)

Field	Description
CFG_EN_PARTIAL	If this bit is asserted, partial power management mode is enabled. When it is de-asserted a PMNACK primitive is sent in response to the reception of the PMREQ_P primitive. Internal power management is always enabled.
CFG_LINK_WAKEUP	When set, this bit causes the SATA controller to wake up from partial/slumber power management mode. It should be kept asserted until PHY_READY is re-asserted in the Phy layer.
CFG_LINK_GO_SLUMBER	When set, this bit causes the SATA controller to request the device to enter slumber power management mode. PMREQ_S primitives is sent to the device.
CFG_LINK_GO_PARTIAL	When set, this bit causes the SATA controller to request the device to enter partial power management mode. PMREQ_P primitives is sent to the device.

Offset 0x0000_0148

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7
conventionl	31	30	29	28	27	26	25	24
R		CFG_EN_PARTIAL	CFG_LINK_WAKEUP	CFG_LINK_GO_SLUMBER	CFG_LINK_GO_PARTIAL	CFG_PRIM_OVR_EN	CFG_PHY_READY_TIMER	
W								
Reset	0	0	0	0	0	0	0	0
Power Architecture	8	9	10	11	12	13	14	15
conventionl	23	22	21	20	19	18	17	16
R	CFG_PHY_READY_TIMER							
W								
Reset	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23
conventionl	15	14	13	12	11	10	9	8
R	CFG_ALIGN_RATE							
W								
Reset	1	1	1	1	1	1	1	1
Power Architecture	24	25	26	27	28	29	30	31
conventionl	7	6	5	4	3	2	1	0
R	CFG_EN_PHY_TO	CFG_SEND_4_ALIGNS	CFG_EN_SLUMBER	CFG_RX_SCRAMBLE	CFG_TX_SCRAMBLE	CFG_TXPRI_M_JUNK	CFG_TX_CONT	CFG_TX_BADCRC
Reset	0	0	0	1	1	0	1	0

= Unimplemented or Reserved

Figure 33-6. Link_cfg0 Register
(Register is repeated for reference.)

Table 33-10. Link_cfg0 Field Descriptions (Sheet 2 of 4)

Field	Description
CFG_PRIM_OVR_EN	Primitive Override Enable. When set, this bit enables the replacement of a single primitive, as specified by CFG_PRIM/CFG_CD, when the link layer state machine is in the CFG_PRIM_OVR_STATE state. This bit has to be toggled from a 0 to a 1 to enable this feature.
CFG_PHY_READY_TIMER	These 10-bits specify the timeout value of the PHY_READY Timer. If CFG_EN_PHY_TO is set, the link layer counts down on every rising edge of scanTxClk, as long as PHY_READY is not ready. When the counter reaches zero, the PHY is reset to try to re-establish communications with the device. The timer is initially loaded with a value equal to the concatenation of {PHY_READY_TIMER, 9 h000}.
CFG_ALIGN_RATE	The SATA specification requires the SATA controller sends a pair of ALIGN primitives at least every 254 doublewords of data. This is achieved by setting CFG_ALIGN_RATE to 11111111. However, for test purposes, it is possible to send ALIGNs at a higher rate. This can be achieved by setting CFG_ALIGN_RATE to a lower value (CFG_ALIGN_RATE-1). Doublewords are sent by the SATA controller between each set of ALIGN primitive pairs. Note: If CFG_SEND_4_ALIGNS is set, do not set the CFG_ALIGN_RATE to be four or less. If CFG_SEND_4_ALIGNS is not set, do not set the CFG_ALIGN_RATE to be two or less.

Offset 0x0000_0148

Access: User read/write

Power	0	1	2	3	4	5	6	7
Architecture								
conventionl	31	30	29	28	27	26	25	24
R		CFG_EN_	CFG_LINK_	CFG_LINK_	CFG_LINK_	CFG_PRIM_	CFG_PHY_READY_TIMER	
W		PARTIAL	WAKEUP	GO_	GO_	OVR_EN		
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventionl	23	22	21	20	19	18	17	16
R	CFG_PHY_READY_TIMER							
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventionl	15	14	13	12	11	10	9	8
R	CFG_ALIGN_RATE							
W								
Reset	1	1	1	1	1	1	1	1
Power	24	25	26	27	28	29	30	31
Architecture								
conventionl	7	6	5	4	3	2	1	0
R	CFG_EN_	CFG_SEND	CFG_EN_	CFG_RX_	CFG_TX_	CFG_TXPRI	CFG_TX_	CFG_TX_
	PHY_TO	_4_ALIGN	SLUMBER	SCRAMBLE	SCRAMBLE	M_JUNK	CONT	BADCRC
Reset	0	0	0	1	1	0	1	0

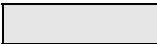
 = Unimplemented or Reserved

Figure 33-6. Link_cfg0 Register
(Register is repeated for reference.)

Table 33-10. Link_cfg0 Field Descriptions (Sheet 3 of 4)

Field	Description
CFG_EN_PHY_TO	If PHY is not ready for a length of time, as specified by CFG_PHY_READY_TIMER, this bit, when asserted, enables to re-issue a reset of the PHY.
CFG_SEND_4_ALIGN	When asserted, four ALIGN primitives are transmitted at the specified rate, instead of the normal two ALIGNs.
CFG_EN_SLUMBER	If this bit is asserted, slumber power management mode is enabled. When it is de-asserted a PMNACK primitive is sent in response to the reception of the PMREQ_S primitive. Internal power management is always enabled.
CFG_RX_SCRAMBLE	If this bit is asserted, de-scrambling of the receive data is enabled as per the SATA specification.
CFG_TX_SCRAMBLE	If this bit is asserted, scrambling of the transmit data is enabled as per the SATA specification.
CFG_TXPRIM_JUNK	If this bit is de-asserted, scrambled junk data is sent after a CONT primitive, as per the SATA specification. If this bit is asserted, the single character 32 hDEADBEEF is sent continuously instead. This is to aid debug.

Offset 0x0000_0148

Access: User read/write

Power	0	1	2	3	4	5	6	7
Architecture								
conventionl	31	30	29	28	27	26	25	24
R		CFG_EN_	CFG_LINK_	CFG_LINK_	CFG_LINK_	CFG_PRIM_	CFG_PHY_READY_TIMER	
W		PARTIAL	WAKEUP	GO_	GO_	OVR_EN		
				SLUMBER	PARTIAL			
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventionl	23	22	21	20	19	18	17	16
R	CFG_PHY_READY_TIMER							
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventionl	15	14	13	12	11	10	9	8
R	CFG_ALIGN_RATE							
W								
Reset	1	1	1	1	1	1	1	1
Power	24	25	26	27	28	29	30	31
Architecture								
conventionl	7	6	5	4	3	2	1	0
R	CFG_EN_	CFG_SEND	CFG_EN_	CFG_RX_	CFG_TX_	CFG_TXPRI	CFG_TX_	CFG_TX_
	PHY_TO	_4_ALIGNS	SLUMBER	SCRAMBLE	SCRAMBLE	M_JUNK	CONT	BADCRC
Reset	0	0	0	1	1	0	1	0

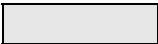
 = Unimplemented or Reserved

Figure 33-6. Link_cfg0 Register
(Register is repeated for reference.)

Table 33-10. Link_cfg0 Field Descriptions (Sheet 4 of 4)

Field	Description
CFG_TX_CONT	If this bit is asserted, the transmission of CONT primitives is enabled. If de-asserted, long sequences of repeated primitives can be sent by the link layer.
CFG_TX_BADCRC	A bad CRC (inverted value of the correct CRC) value is transmitted for one FIS only by the link SATA controller rising edge is detected on this signal. This bit has to be toggled from a 0 to a 1 to enable this feature.

33.3.1.3.4 Link_cfg1 Register

Offset 0x0000_014c				Access: User read/write				
Power	0	1	2	3	4	5	6	7
Architecture								
conventional	31	30	29	28	27	26	25	24
R								
W								
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventional	23	22	21	20	19	18	17	16
R								
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventional	15	14	13	12	11	10	9	8
R								
W								
Reset	0	0	0	0	0	0	0	0
Power	24	25	26	27	28	29	30	31
Architecture								
conventional	7	6	5	4	3	2	1	0
R	CFG_RX_BADCRC	CFG_CD	CFG_PRIM_OVR_STATE					
W								
Reset	0	0	0	0	0	0	0	0
		= Unimplemented or Reserved						

Figure 33-7. Link_cfg1 Register

Table 33-11. Link_cfg1 Field Descriptions

Field	Description
CFG_RX_BADCRC	When a rising edge is detected on this bit, it causes a bad CRC to be detected for the current frame. This bit has to be toggled from a 0 to a 1 to enable this feature.
CFG_CD	This bit specifies whether the data used during the primitive override should be a data character or a primitive. If CFG_CD equals 1, CFG_PRIM_OVR_STATE equals L_SendEOF, and CFG_PRIM equals WTRM, a WTRM primitive is inserted into the datastream instead of an EOF (when a rising edge is seen on CFG_PRIM_OVR_EN). If CFG_CD equals 0, a normal data character (as specified by CFG_PRIM) is inserted into the datastream instead of the EOF.
CFG_PRIM_OVR_STATE	These six bits are used in the primitive override debug functionality. When the SATA controller detects a positive edge on CFG_PRIM_OVR_EN, it overrides the next primitive that would be inserted during the CFG_PRIM_OVR_STATE, with the data specified by the CFG_PRIM and CFG_CD configuration bits.

33.3.1.3.5 Link_cfg2 Register

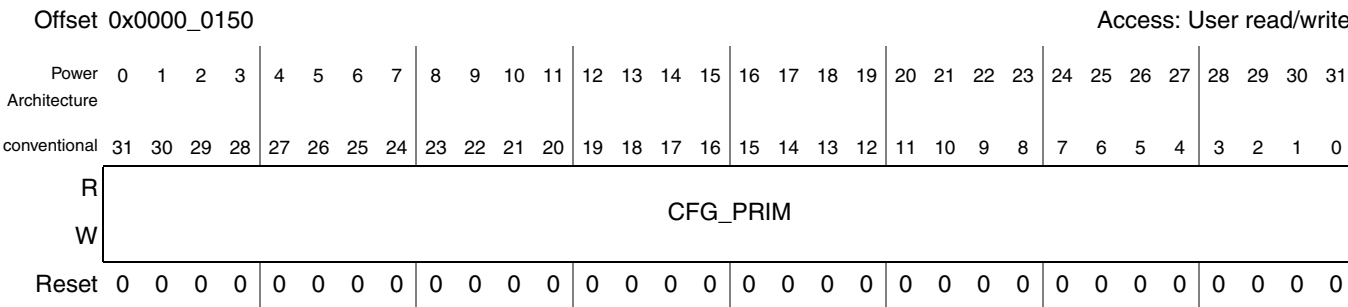


Figure 33-8. Link_cfg2 Register

Table 33-12. Link_cfg2 Field Descriptions

Field	Description
CFG_PRIM	This 32-bit bus specifies the data to be used in the overriding primitive debug logic, described in the definition of LINK_CFG1 above.

33.3.1.3.6 Link_status0 Register

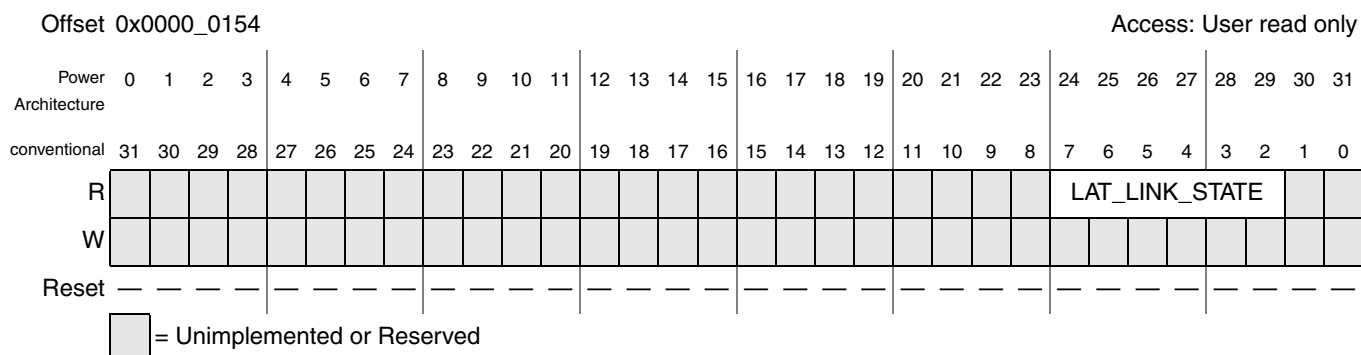


Figure 33-9. Link_status0 Register

Table 33-13. Link_status0 Field Descriptions

Field	Description
LAT_LINK_STATE	<p>These six bits specify the current value of the Link Layer State Machine at the time the Status0 register is read.</p> <p>0 : L_Reset ; 22:L_WakeUp1 ; 1 : L_Idle ; 23:L_WakeUp2 ; 2 : HL_SendChkRdy ; 24:L_RcvChkRdy ; 3 : DL_SendChkRdy ; 25:L_RcvData ; 4 : L_TPMPartial ; 26:L_BadEnd ; 5 : L_TPMSlumber ; 27:L_RcvEOF ; 6 : L_RcvWaitFIFO ; 28:L_SendHoldA ; 7 : L_PMOff ; 29:L_Hold ; 8 : L_PMDeny ; 30:L_GoodCRC ; 9 : L_NoCommErr ; 31:L_GoodEnd ; 10:L_NoComm ; 32:L_PMOff_2 ; 11:L_SendAlign ; 33:L_PMOff_3 ; 12:L_SendSOF ; 34:L_PMOff_4 ; 13:L_SendData ; 35:WAIT_PMACK_SENT_1 ; 14:WAIT_FOR_SYNC ; 36:WAIT_PMACK_SENT_2 ; 15:L_SendCRC ; 37:WAIT_PMACK_SENT_3 ; 16:L_SendHold ; 38:WAIT_PMACK_SENT_4 ; 17:L_RcvHold ; 39:WAIT_PMACK_SENT_5 ; 18:L_SendEOF ; 40:WAIT_PMACK_SENT_6 ; 19:L_Wait ; 41:BIST0 ; 20:L_ChkPhyRdy ; 42:BIST1 ; 21:L_NoCommPower ;</p> <p>Consult section 7.6 in SATA specification (Revision 1.0a) for more details about link layer state machine.</p>

33.3.1.3.7 Link_status1 Register

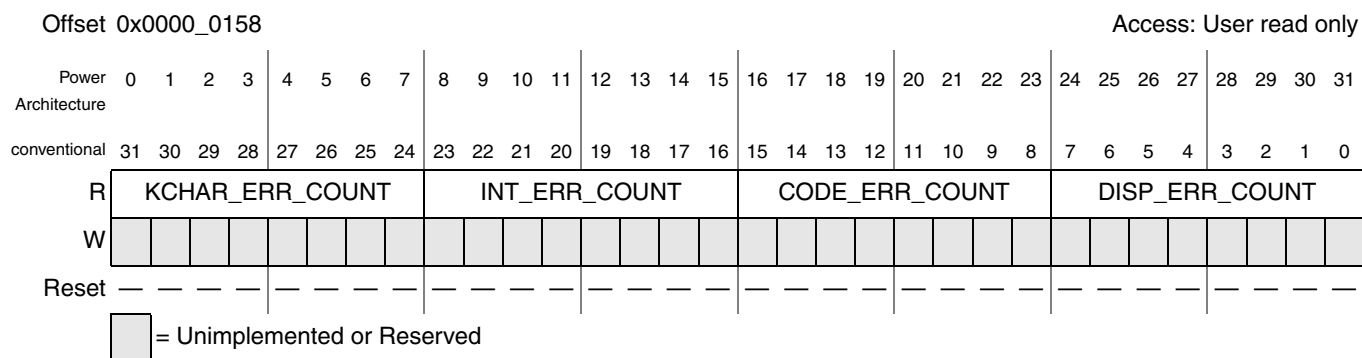


Figure 33-10. Link_status1 Register

Table 33-14. Link_status1 Field Descriptions

Field	Description
KCHAR_ERR_COUNT	The number of doublewords that have been received from the PHY, where one or more control character errors have been detected. A value of 11111111 indicates an error count of 255 or more as this counter does not wrap around to zero. The count value is updated with its current value each time the LINK_STATUS1 register is read.
INT_ERR_COUNT	The number of doublewords that have been received from the PHY, where one or more internal errors have been detected. A value of 11111111 indicates an error count of 255 or more as this counter does not wrap around to zero. The count value is updated with its current value each time the LINK_STATUS1 register is read.
CODE_ERR_COUNT	The number of doublewords that have been received from the PHY, where one or more code errors have been detected. A value of 11111111 indicates an error count of 255 or more as this counter does not wrap around to zero. The count value is updated with its current value each time the LINK_STATUS1 register is read.
DISP_ERR_COUNT	The number of doublewords that have been received from the PHY, where one or more disparity errors have been detected. A value of 11111111 indicates an error count of 255 or more as this counter does not wrap around to zero. The count value is updated with its current value each time the LINK_STATUS1 register is read.

33.3.1.3.8 Phy_cfg Register

Offset 0x0000_0174

Access: User read/write

Power	0	1	2	3	4	5	6	7
Architecture								
conventional	31	30	29	28	27	26	25	24
R								
W								
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventional	23	22	21	20	19	18	17	16
R							CFG_TX_PRE_EM_CTRL:	
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventional	15	14	13	12	11	10	9	8
R	CFG_TX_PRE_EM_CTRL	CFG_TX_AMP_CTRL				CFG_ENDEC	CFG_RX_SSC_MODE	CFG_TX_SSC_MODE
W								
Reset	0	0	1	1	1	1	1	0
Power	24	25	26	27	28	29	30	31
Architecture								
conventional	7	6	5	4	3	2	1	0
R	CFG_TX_SSC_MODE		CFG_FRC_PHYRDY	CFG_RXREFMODE	ENLWALIGN	CFG_LBEN		CFG_BIST
W								
Reset	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 33-11. Phy_cfg Register

Table 33-15. Phy_cfg Field Descriptions

Field	Description
CFG_TX_PRE_EM_CTRL	Drives TxPreEmCtrl of Phy when prot2PhyTstEnL equals 0.
CFG_TX_AMP_CTRL	Drives TxAmpCtrl of Phy when prot2PhyTstEnL equals 0.
CFG_ENDEC	Drives enDec of PCS when prot2PhyTstEnL equals 0.
CFG_RX_SSC_MODE	Drives RxSSCMode of Phy when prot2PhyTstEnL equals 0.
CFG_TX_SSC_MODE	Drives TxSSCMode of PCS when prot2PhyTstEnL equals 0.
CFG_FRC_PHYRDY	Drives frcPhyRdy of PCS when prot2PhyTstEnL equals 0.
CFG_RXREFMODE	Drives rxRefMode of PCS when prot2PhyTstEnL equals 0.
ENLWALIGN	Enable Longword Alignment. The Phy control layer decides whether to align the control byte of a SATA primitive to the lowest byte position of the doublewords.

Table 33-15. Phy_cfg Field Descriptions (continued)

Field	Description
CFG_LBEN	Drives lbEn of PCS when prot2PhyTstEnL equals 0.
CFG_BIST	Drives bist of PCS when prot2PhyTstEnL equals 0.

33.3.1.3.9 Trans_status1 Register

Offset 0x0000_0178

Access: User read only

Power	0	1	2	3	4	5	6	7
Architecture								
conventional	31	30	29	28	27	26	25	24
R								
W								
Reset	—	—	—	—	—	—	—	—
Power	8	9	10	11	12	13	14	15
Architecture								
conventional	23	22	21	20	19	18	17	16
R							RX_FIFO_OVERFLOW	TX_FIFO_EMPTY
W								
Reset	—	—	—	—	—	—	—	—
Power	16	17	18	19	20	21	22	23
Architecture								
conventional	15	14	13	12	11	10	9	8
R	CUR_TRANS_VALUE							
W								
Reset	—	—	—	—	—	—	—	—
Power	24	25	26	27	28	29	30	31
Architecture								
conventional	7	6	5	4	3	2	1	0
R	CUR_TRANS_VALUE							
W								
Reset	—	—	—	—	—	—	—	—

= Unimplemented or Reserved

Figure 33-12. Trans_status1 Register

Table 33-16. Trans_status1 Field Descriptions

Field	Description
RX_FIFO_OVERFLOW	This bit is set to one by hardware if receive FIFO is overflowed.

Table 33-16. Trans_status1 Field Descriptions (continued)

Field	Description
TX_FIFO_EMPTY	This bit is set to one if transmit FIFO is empty.
CUR_TRANS_VALUE	Current value of transfer count.

33.3.1.3.10 Tx_bist_dat0 Register

Offset 0x0000_017c

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TX_BIST_DAT0																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 33-13. Tx_bist_dat0 Register

Table 33-17. Tx_bist_dat0 Field Descriptions

Field	Description
TX_BIST_DAT0	BIST DATA 0 used in BIST mode.

33.3.1.3.11 Bist_dat0_sel Register

Offset 0x0000_0180

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																																S
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 33-14. Bist_dat0_sel Register

Table 33-18. Bist_dat0_sel Field Descriptions

Field	Description
S	0 Indicates the value in TX_BIST_DAT0 register is a data. 1 Indicates the value in TX_BIST_DAT0 register is a primitive.

33.3.1.3.12 Tx_bist_dat1 Register

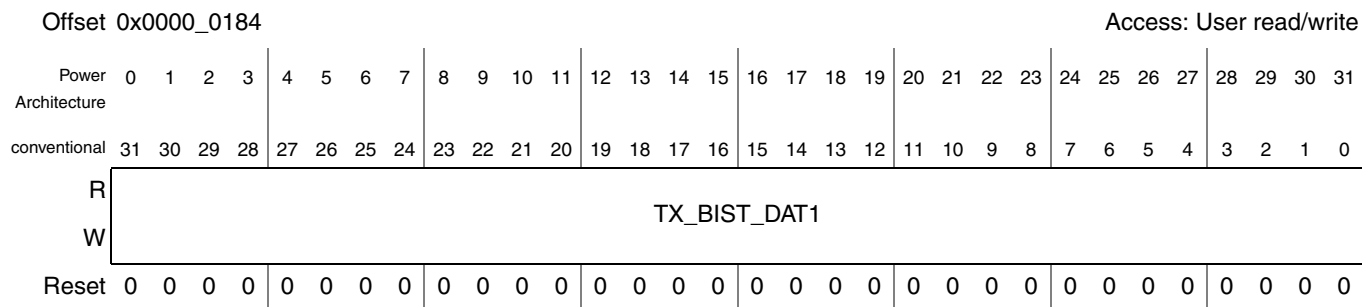


Figure 33-15. Tx_bist_dat1 Register

Table 33-19. Tx_bist_dat1 Field Descriptions

Field	Description
TX_BIST_DAT1	BIST DATA 1 used in BIST mode.

33.3.1.3.13 Bist_dat1_sel Register

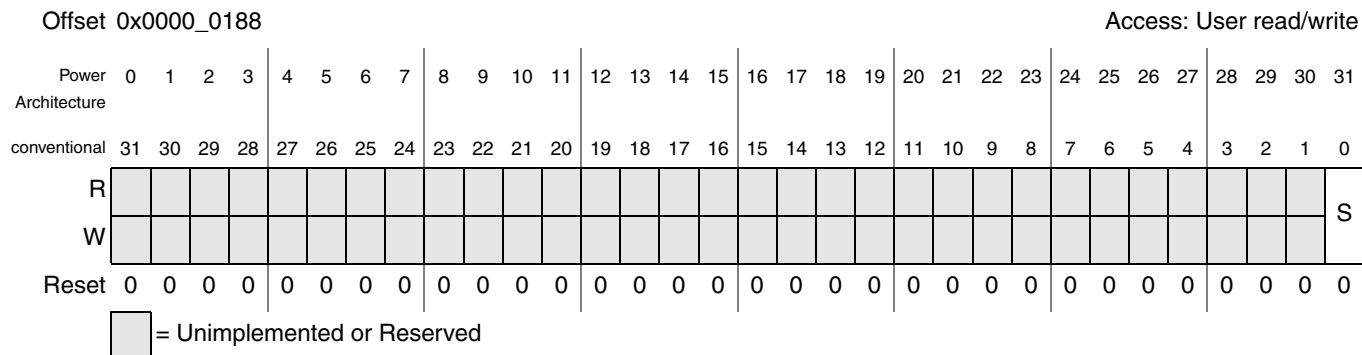


Figure 33-16. Bist_dat1_sel Register

Table 33-20. Bist_dat1_sel Field Descriptions

Field	Description
S	0 Indicates the value in TX_BIST_DAT1 register is a data. 1 Indicates the value in TX_BIST_DAT1 register is a primitive.

33.3.1.3.14 Tx_bist_mode Register

The value of this register (only eight valid bits) are put in the host-to-device BIST activate FIS as pattern definition field.

See Section 8.5.7 in SATA specification (Revision 1.0a) for more details.

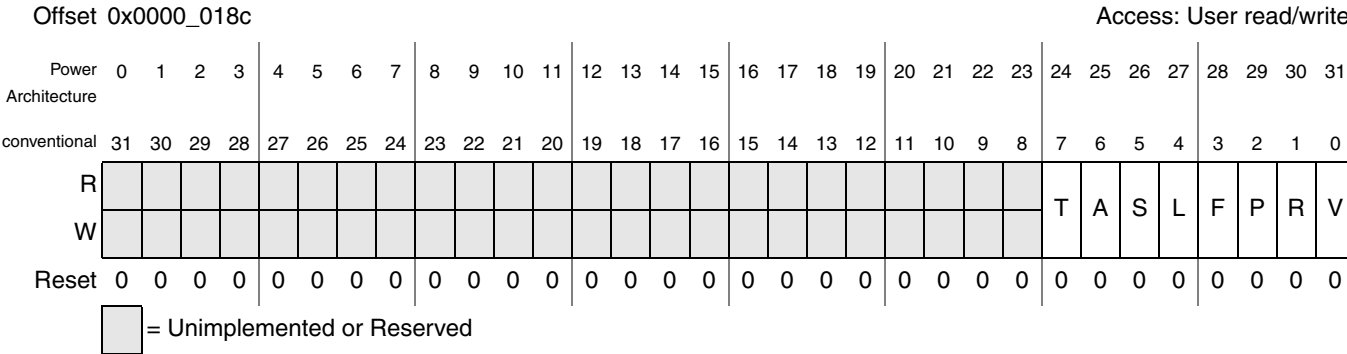


Figure 33-17. Tx_bist_mode Register

Table 33-21. Tx_bist_mode Field Descriptions

Field	Description
T	Device transmit only mode.
A	ALIGN Bypass (Do not transmit ALIGN Primitives) (valid only in combination with T bit)
S	Bypass scrambling (valid only in combination with T bit) (optional behavior)
L	Device Retimed Loopback. Transmitter must insert additional ALIGNs.
F	Device Analog Loopback. (optional)
P	Primitive bit. Valid only in combination with T bit. Optional behavior.
R	Reserved, 0.
V	Vendor unique Test Mode. Causes all other bits to be ignored.

33.3.1.3.15 Bist_written Register

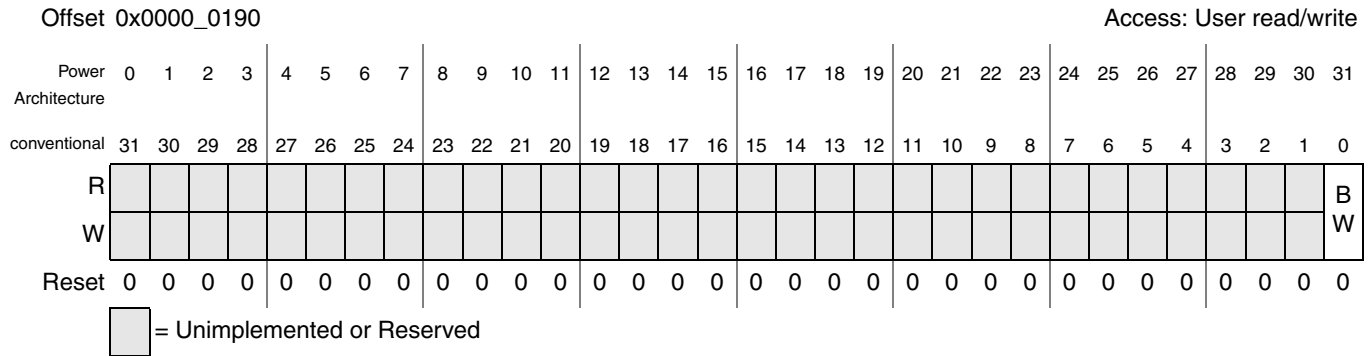


Figure 33-18. Bist_written Register

Table 33-22. Bist_written Field Descriptions

Field	Description
BW	Writes 1 to this bit triggering a host initiated BIST.

33.3.1.3.16 Bist_status Register

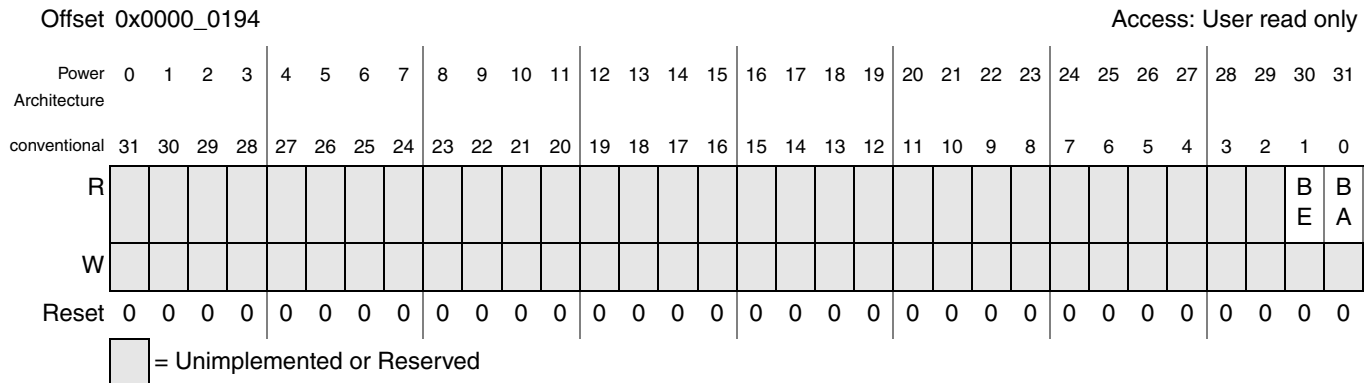


Figure 33-19. Bist_status Register

Table 33-23. Bist_status Field Descriptions

Field	Description
BE	0 No Bist error. 1 Bist error
BA	0 Bist is inactive 1 Bist is active.

33.3.1.3.17 Tx_send_dmat Register

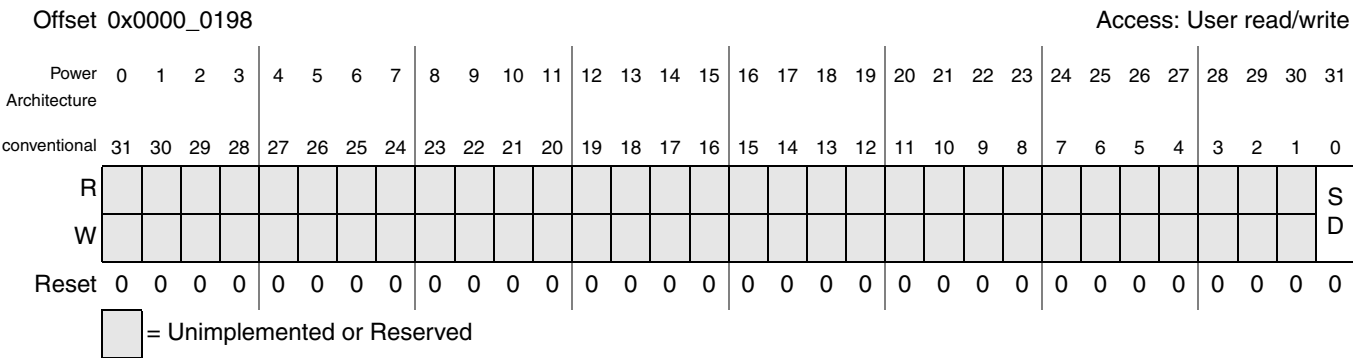


Figure 33-20. Tx_send_dmat Register

Table 33-24. Tx_send_dmat Field Descriptions

Field	Description
SD	SATA controller sends a DMAT primitive when this bit is set to one.

33.3.1.3.18 Atapi_en Register

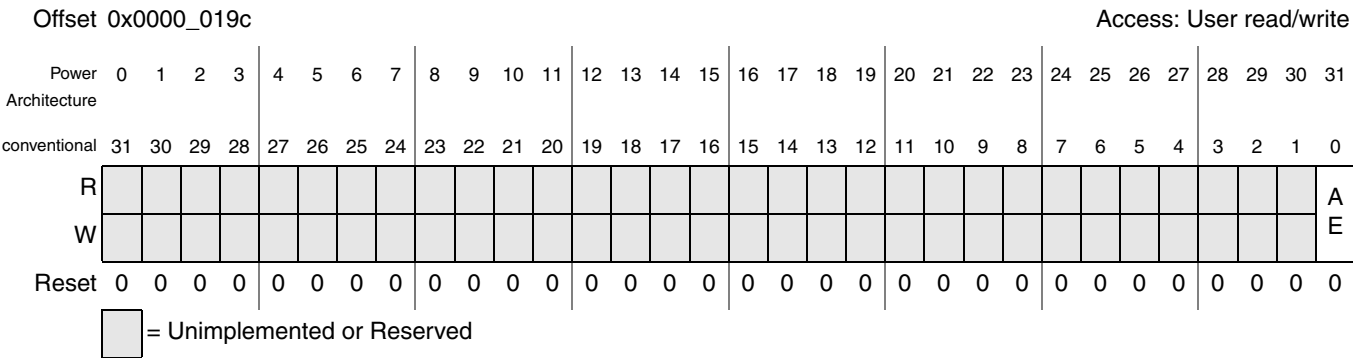


Figure 33-21. Atapi_en Register

Table 33-25. Atapi_en Field Descriptions

Field	Description
AE	0 Disable ATAPI mode 1 Enable ATAPI mode.

33.3.1.4 Additional Registers

33.3.1.4.1 PMCtrl Register

Offset 0x0000_01d4												Access: User read/write				
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									LBENASEL	LBENA_FSL			SATA_CTL_EN	SATA_PCS_EN		PHY_ENABLE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved															

Figure 33-22. PMCtrl Register

Table 33-26. PMCtrl Field Descriptions

Field	Description
LBENASEL	0 Select default valule from controller to control loopback mode in SATA PHY. 1 Select LBENA_FSL in this register to control loopback mode in SATA PHY.
LBENA_FSL	Provides additional working mode for Freescale SATA PHY. 000 Normal function mode 001 Far-end retimed parallel loopback (PCS Rx->Tx) 010 Internal serial loopback (PHY Tx->Rx) 011 Disable digital reset during debug&invalid loopback 100 Internal serial loopback (receiver -> transmitter) 101 goParallel, ORs into intGoParOrSlumA 110 goSlumber, ORs into intGoParOrSlum 111 Invalid
SATA_CTL_EN	This bit is sata controller enable. It can be set to '1' if sata pcs has been enabled(Referring to Section Initialization Information for details). 0 Disable sata controller. 1 Enable sata controller.
SATA_PCS_EN	This bit is sata pcs enable. It can be set to '1' if sata phy PLL has been locked.(Referring to Section Initialization Information for details). 0 Disable SATA PCS. 1 Enable SATA PCS.
PHY_ENABLE	0 Shut down SATA PHY and related oscillator. 1 Enable SATA PHY and related oscillator.

33.3.1.4.2 PMStat Register

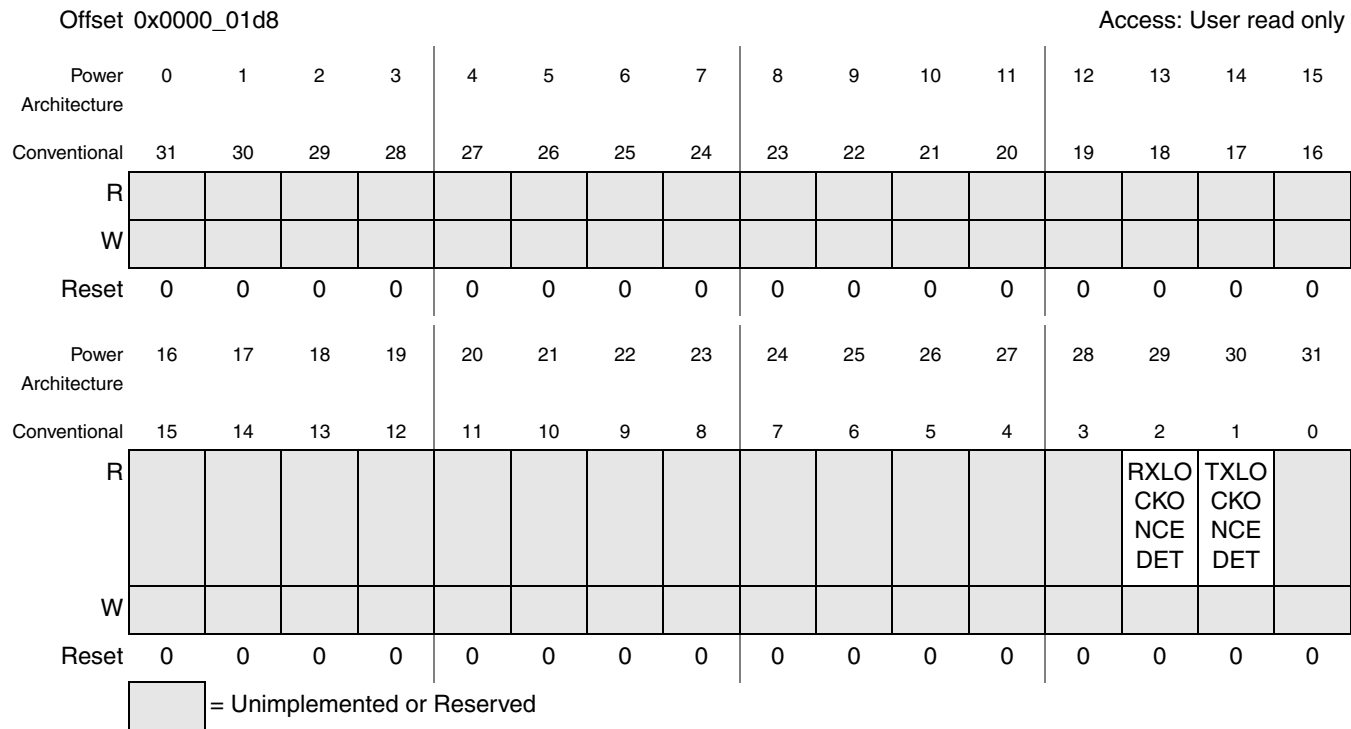


Figure 33-23. PMStat Register

Table 33-27. PMStat Field Descriptions

Field	Description
RXLOCKONCEDET	Receiver PLL lock once detect.
TXLOCKONCEDET	Transmitter PLL lock once detect.

33.3.1.5 DMA Registers

33.3.1.5.1 DmaEndian Register

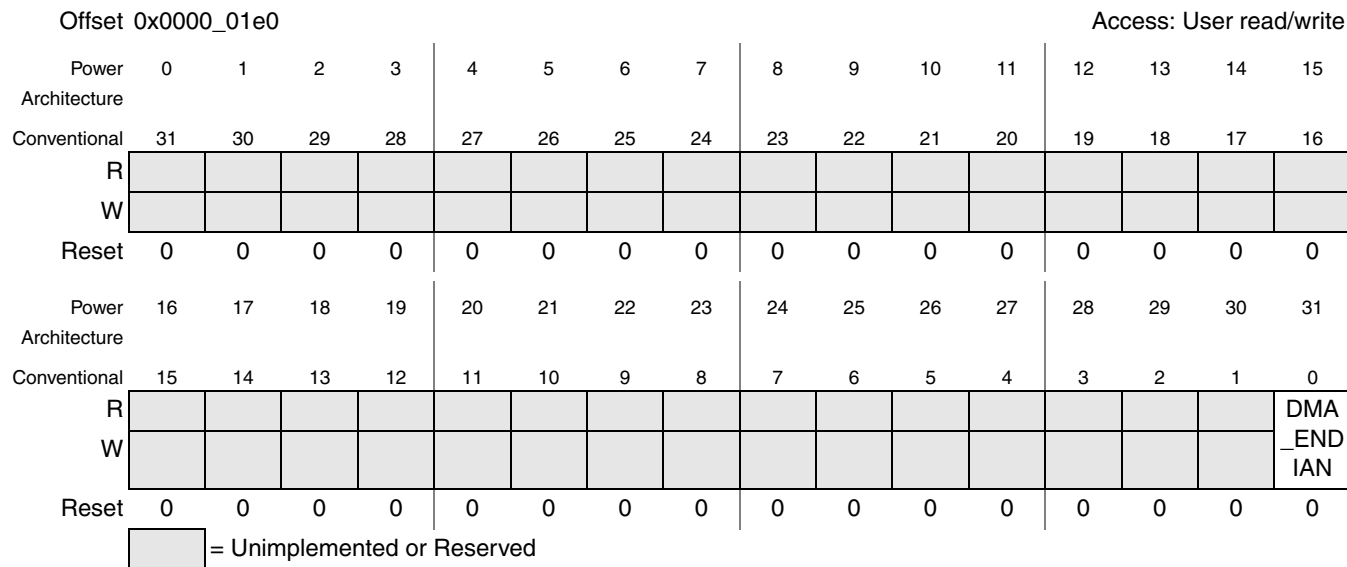


Figure 33-24. DmaEndian Register

Table 33-28. Register DmaEndian Field Descriptions

Field	Description
DMA_ENDIAN	0 DMA interface is big-endian. 1 DMA interface is little-endian

33.3.1.5.2 Dma_stat_ctrl Register

Offset 0x0000_1080

Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R		RDPNTRA			RDPNTRB				WRPNTRA				WRPNTRB				BUFSTAT		STAL LINT
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R														STAL LINT EN	DISK READ	DMA EN			
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0			

= Unimplemented or Reserved

Figure 33-25. Dma_stat_ctrl Register

Table 33-29. Dma_stat_ctrl Field Descriptions

Field	Description
RDPNTRA	Read pointer for FIFO A
RDPNTRB	Read pointer for FIFO B
WRPNTRA	Write pointer for FIFO A
WRPNTRB	Write pointer for FIFO B
BUFSTAT	DMA Buffer status: 00 DMA is disabled. 01 DMA is stalled. 10 DMA is on progress. 11 Reserved.
DISKREAD	Indicates disk read or disk write operation. 0 This is a disk write operation. DMA controller reads data from system memory. 1 This is a disk read operation. DMA controller writes data to system memory.
DMAEN	DMA enable. 0 DMA is disabled. 1 DMA is enabled.

33.3.1.5.3 Dma_max_cnt Register

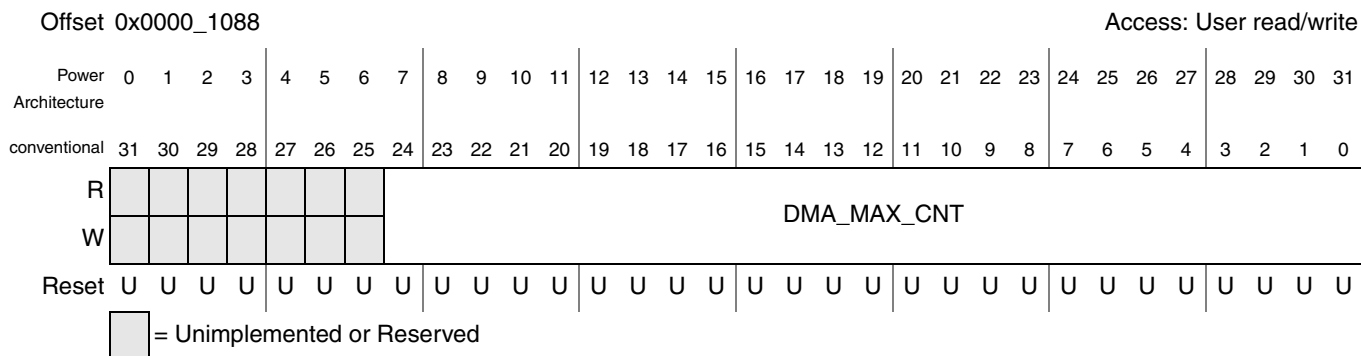


Figure 33-26. Dma_max_cnt Register

Table 33-30. Register Dma_max_cnt Field Descriptions

Field	Description
DMA_MAX_CNT	Maximum size (doublewords) for DMA transferring. No reset value of this register.

33.3.1.5.4 Dma_start Register

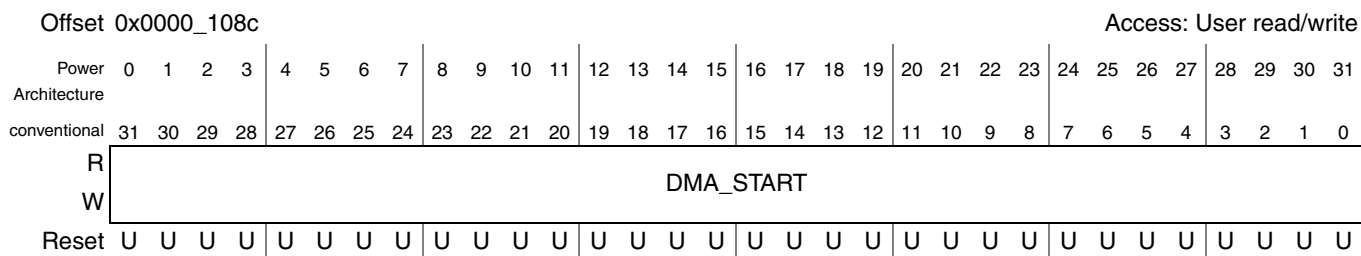


Figure 33-27. Dma_start Register

Table 33-31. Dma_start Field Descriptions

Field	Description
DMA_START	Start address (system memory address) for DMA transferring. No reset value of this register.

33.4 Functional Description

33.4.1 Clock

There are two clock sources for SATA block. One is clock generation block. The other is PLL in SATA PHY. Please refer to SATA related sections in block guide of clock generation module.

A PLL is embeded in SATA PHY. This PLL provides clocks not only for SATA PHY itself, but also for SATA controller. While before clocks from SATA PHY are stable, SATA controller should stay in reset to avoid wrong logic. User should wait for some time after enabling SATA PHY, then set SATA_PCS_EN and SATA_CTL_EN of PMCTRL register to start any operations.

33.4.2 Interrupt

There are two interrupt signals from SATA block. One is SATA_CMD_INTRQ from Command Layer. The other is SATA_DMA_INTRQ from DMA controller. A dedicated Interrupt output named SATA_CMD_INTRQ from the command layer to the host. This interrupt is asserted if the IPF (interrupt pending flag) is set in the command layer and the interrupt enable bit is low, NIEN (bit[1]), of the device control register (address offset 0x0000_0048). The default value of NIEN is 1'b1; interrupts are disabled. The command layer clears the interrupt pending flag any time the status register is read with DEV cleared to zero, the SRST bit in the device control register is set to one, the command register is written with DEV cleared to zero, or a COMRESET is requested by the host to be sent to the device.

A dedicated interrupt output (SATA_DMA_INTRQ) from the DMA controller to the host system. This interrupt is asserted if the StallInt (Stall State Interrupt) is set in the DMA controller and the interrupt enable bit is high bit 2 of the DMA_STAT_CTRL register. The StallInt generated within the DMA controller indicates registers are available for updating due to transfer completion. The host system can clear the interrupt by writing new register values to begin a new transfer or disabling the DMA controller.

33.4.3 ATAPI Support

Host software must write a 1 to the AE in ATAPI_EN register when it has determined that the device is an ATAPI device (via the signature returned from the device).

If AE is asserted and the host reads any taskfile register except device control or status while the DEV bit set, the command layer returns all zeros on register access bus as per the ATA/ATAPI specification.

The host can issue a device reset command at any time, regardless of the status of the BSY bit. When the command register is written to with a value of 8'h08, SATA controller transmit a device reset command to the device. When the command has been successfully transmitted, the device reset pending signal is cleared automatically.

33.4.3.1 ATAPI Transfers

Refer to Appendix D in SATA specification (Revision 1.0a) for the detailed process of ATAPI transfers.

33.4.4 PIO Transfers

Programmed I/O (PIO) transfers are implemented using the data register (address offset 0x0000_0000) in the ATA taskfile and control register.

Refer to Appendix D in SATA specification (Revision 1.0a) for the detailed process of ATAPI transfers.

33.4.4.1 PIO Read

A PIO data read consists of the following sequence of events:

1. Host software writes to the taskfile command register with a PIO read command (READ SECTOR).
2. The SATA controller sends a Register Update FIS (Host-to-Device) to the device.
3. The device sends a PIO Setup FIS back to the host.

4. The device sends data to the host.
5. When the data is in the buffer of SATA controller from device, the DRQ (Data Request) bits is set in the taskfile status register and host system read data from taskfile data register.
6. Keep reading taskfile data register until no more data is required. DRQ is then de-asserted.

33.4.4.2 PIO Write

A PIO data write consists of the following sequence of events:

1. Host software writes to the taskfile command register with a PIO write command (WRITE SECTOR).
2. The SATA controller sets the DRQ status bit and the host software writes the first data into the taskfile data register.
3. SATA controller transfer data to device.
4. Host software keeps writing data until no more data is required. DRQ is then de-asserted.

33.4.5 DMA Transfers

DMA transfers are implemented using the registers in command layer outlined previously. Refer to Appendix D in SATA specification (Revision 1.0a) for the detailed process of ATAPI transfers.

33.4.5.1 DMA Read

A DMA data read from the device consists of the following sequence of events:

1. Host software writes to DMA_STAT_CTRL register to setup and enable a READ_DMA.
2. Host software writes to DMA_MAX_CNT register the number of doublewords transfers requested.
3. Host software writes to DMA_START register the initial start address of the host system memory.
4. The host software writes the READ_DMA command to the taskfile command register.
5. DMA transfer begins and data is transfered to system memory via data transfer interface.
6. When the DMA_REMAIN_AHB register reaches zero, the READ_DMA transfer has completed and the StallInt is set in the DMA_STAT_CTRL register. A hardware interrupt SATA_DMA_INTRQ is asserted if the StallIntEn bit has been set in the DMA_STAT_CTRL register. StallInt is cleared the next time the host writes to the start address register or if the DMA engine is disabled in the DMA_STAT_CTRL register.

33.4.5.2 DMA Write

A DMA data write to the device consists of the following sequence of events:

1. Host software writes the WRITE_DMA command to the taskfile command register.
2. Host software writes to DMA_STAT_CTRL register to setup and enable a WRITE_DMA.
3. Host software writes to DMA_MAX_CNT register the number of doublewords transfers requested.
4. Host software writes to DMA_START register the initial start address of the system memory.

5. This sequence initiates the DMA Controller requesting data from system memory to its internal buffer, then data is transferred to device.
6. When the DMA_REMAIN_TC register reaches zero, the DMA controller asserts the DMA_TX_TC signal. When the WRITE_DMA transfer has completed, the STALLINTEN bit is set in the DMA_STAT_CTRL register. STALLINT is cleared the next time the host writes to the start address register or if the DMA engine is disabled in the DMA_STAT_CTRL register.

The order of command issuing and DMA controller programming can be altered or reversed from the above examples. However, the DMA_START register must always be the last register to be programmed in the DMA controller because this starts the transfer.

33.4.6 Power Management

A power management logic is designed for managing change of power mode requests. These requests can come from the SATA controller or the device. The host software can change the power mode by asserting CFG_LINK_GO_PARTIAL or CFG_LINK_GO_SLUMBER bits in the Link_cfg0 register. Asserting the SATA controller transmits PMREQ_P (CFG_LINK_GO_PARTIAL = 1) or PMREQ_S (CFG_LINK_GO_SLUMBER = 1) primitives to the device and waits for PMACK primitives from it in response. After PMACK is received, the SATA controller enters the partial or slumber power down state.

NOTE

CFG_LINK_GO_PARTIAL or CFG_LINK_GO_SLUMBER must remain assert for the duration of the power down mode.

A write of 1 to the CFG_LINK_WAKEUP bit in the Link_cfg0 register or reception of a COMWAKE from the device initiates a resume to active power mode. CFG_LINK_WAKEUP should be set back to 0 after Phyrdy is asserted again.

If the controller layer receives a PMREQ_P/PMREQ_S primitive from a device and is enabled to perform power management modes (CFG_EN_PARTIAL/SLUMBER = 1 in Link_cfg0 register), it responds by sending at least four PMACK primitives, and enters the requested power down mode (depending on received primitive). A write of 1 to the CFG_LINK_WAKEUP bit or reception of a COMWAKE from the device initiates a resume to active power mode. CFG_LINK_WAKEUP should be set back to 0 after Phyrdy is asserted again.

If the SATA controller receives an XRDY primitive from the device while it is in the partial or slumber state, it returns to idle and signals a link sequence error to the command layer, LINK_SEQ_ERROR equals 1.

33.4.7 DMA Controller

The function of the SATA DMA controller is to provide a high-speed interface between the command/transport layers and system memory.

33.4.7.0.1 DMA Transfers

Refer to [Section 33.4.5, “DMA Transfers”](#)

33.4.7.1 DMA Interrupt Generation

Refer to [Section 33.4.2, “Interrupt](#)

33.4.7.2 Register Block

Register groups implemented in command layer are:

- ATA taskfile and control registers
- SATA status and control registers
- Special registers
- Additional registers

See [Section 33.3, “Memory Map and Register Definition,”](#) for the details.

33.4.7.3 PIO Transfers

Refer to [Section 33.4.4, “PIO Transfers.](#)

33.4.7.4 Interrupt Generation

assumed to be associated with the lowest byte position of the transmit doubleword) is also passed onto the Phy with the appropriate byte.

33.4.8 Physical Coding Sublayer (PCS)

The function of the SATA PCS layer is to interface between the link/PhyCtrl layer on the protocol side and the high speed SerDes (Serializer/Deserializer). It includes OOB (Out Of Band) processor, transmit/receive path control, 8B/10B encoder/decoder, and test function.

33.4.9 Serial ATA Physical Layer Macro (SATA PHY)

33.4.9.1 Introduction

[Figure 33-28](#) shows the top level block diagram of the SATA-I PHY.

The transmitter path starts with a 10-bit symbol sent to the input of the SATA-I PHY at 150MHz. The symbol is serialized and then passed through the data latch at 1.5GHz to reshape the data. This serial data stream is then passed through the pre-driver to the differential driver where the serial bits are converted to a differential signal and sent across the SATA link to the device at a 1.5GHz rate.

The receiver path starts with the differential receiver converting the differential signal received from the device over SATA link to a single-end 1.5GHz serial data stream. This serial data stream is then passed through the CDR block which reconstructs the transmitter clock to reliably sample the data stream. The sampled signals are then sent to the deserializer which creates 10-bit symbols which are sent to the controller at 150MHz. The deserializer aligns these symbols to the comma align sequence contained in the K28.5 control character of the ALIGN primitive.

The transmitter and receiver clock is derived from a common 1.5GHz seven phase PLL. These seven phases provide seven possible points to sample the received data within the 1.5GHz clock period. The PLL synthesizes the 1.5GHz clock from a 25MHz reference frequency.

Two loopback paths are provided in the PHY to help debug the SATA-I system. The near-end loopback from the serializer to the CDR is used to debug the interface between the controller and the PHY. The far-end loopback path from the receiver to the transmit driver is used to debug the device interface with this host PHY.

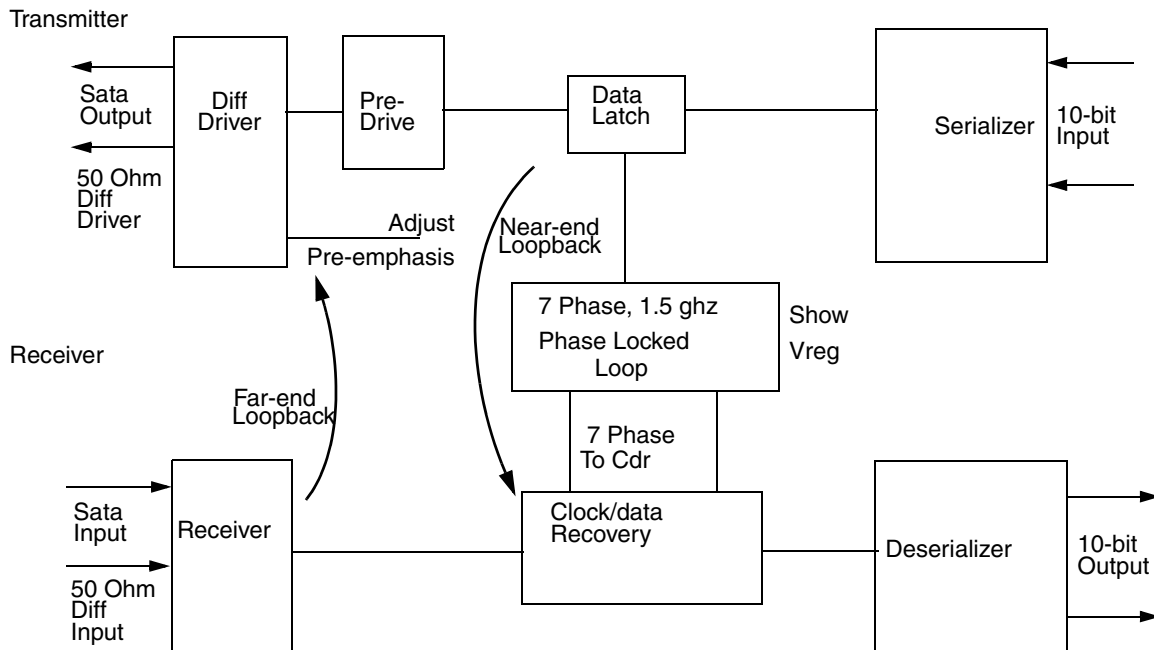


Figure 33-28. SATA-I Host PHY Block Diagram

33.5 Initialization Information

The sequence for sata intialization is shown below.

1. Enable sata phy and reference clock for sata phy PLL.(Set bit phy_enable in PMCtrl register).
2. Wait until sata phy PLL is locked.(Check bit rxLockOnceDet and txLockOnceDet in PMStat register).
3. Enable sata pcs.(Set bit sata_pcs_en in PMCtrl register).
4. Enable sata controller.(Set bit sata_ctl_en in PMCtrl register).

Chapter 34

Secure Digital Host Controller (SDHC)

34.1 Introduction

The Secure Digital Host Controller module (SDHC) controls the Multimedia Card (MMC), Secure Digital (SD) memory card, and I/O cards by sending commands to cards and performing data accesses to/from the cards.

The MMC is a universal low-cost data storage and communication media that covers a wide area of applications such as electronic toys, organizers, PDAs, and smart phones. The MMC communication protocol is based on an advanced 7-pin serial bus that operates in a low-voltage range.

The SD is an evolution of MMC technology with two additional pins in the form factor. SD is specifically designed to meet the security, capacity, performance, and environment requirement inherent in newly emerging audio and video consumer electronic devices. The physical form factor, pin assignment, and data transfer protocol are forward compatible with the MMC with some additions. The memory card invokes a copyright protection mechanism that complies with the SDMI security standard. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices.

The MMC/SD Host module integrates both MMC support along with SD memory and I/O functions.

Figure 34-1 is the system interconnection with this module.

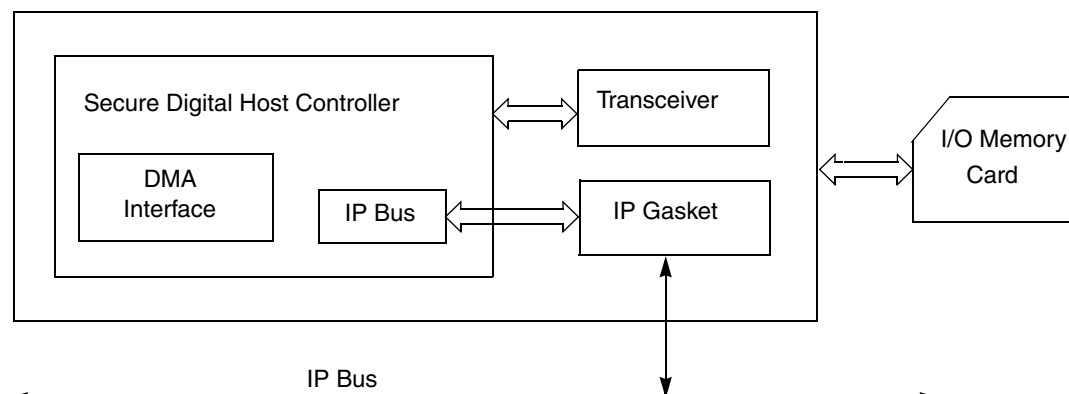


Figure 34-1. System Interconnection with the Secure Digital Host Controller

34.1.1 Features

The features of the Secure Digital Host Controller module include:

- Fully compatible with the MMC System Specification Version 3.2, supports up to version 4.0
- Compatible with high speed MMC card of with using a 1-bit or 4-bit serial interface. An 8-bit interface is not supported
- Compatible with the SDIO Standard and SD Physical layer specification with 1 or 4 channel(s)
- Built-in programmable frequency counter for SDHC bus
- Maskable hardware interrupt for SDIO interrupt, internal status & FIFO status
- Dual data FIFO buffer built-in
- Plug and play (PnP) support
- Single/multi block access to the card including erase operation
- Multi-SD function support including multiple I/O and combined I/O and memory
- Up to 7 I/O functions plus one memory supported on single SD I/O card (Combo card)
- Support for interrupt via IRQ
- Support SDIO Interrupt detection during 1 or 4-bit access
- Block based data transfer between MMC card and SDHC (stream mode or SPI not supported)
- Block length of data transfer between Host and Card can be configured from 1 – 2048 bytes.

34.2 External Signal Description

Table 34-1. Signal Properties

Name	Port	Function	I/O	Reset	Pull up
MMC_SD_CLK	O	Clock for MMC/SD/SDIO card		0	
CMD	I/O	CMD line connect to card		1	Pull up
DAT3	I/O	Card Detect in Power up Data line in 4-bit mode Not used in 1-bit mode		0	Pull down DAT3 if need card detection through this bit, otherwise pull up
DAT2	I/O	Data line or Read wait in 4-bit mode Read wait in 1-bit mode		1	Pull up
DAT1	I/O	Data line or interrupt in 4-bit mode Interrupt in 1-bit mode		1	Pull up
DAT0	I/O	Data line both in 1-bit and 4-bit mode		1	Pull up

34.2.1 Detailed Signal Descriptions

34.2.1.1 Overview

The SDHC has six external pins. MMC_SD_CLK is an internally generated clock used by the MMC/SD Card. The CMD I/O sends commands and receive responses from the card. Four data lines, DAT3~DAT0, perform data transfers between the host controller and the card.

Table 34-2 shows the signal properties of these external pins.

Table 34-2 is a detailed description of the SDHC bus signals.

Table 34-2. Detailed Signal Descriptions

Signal	Descriptions
MMC_SD_CLK	This signal is the MMC/SD/SDIO card clock signal. The direction is from host to card. The frequency is referenced to the system clock.
CMD	This is a bidirectional signal. It is for card initialization and data transfer commands.
DAT3	This is a bidirectional signal for data transmission in 4-bit mode. This signal is not used in 1-bit mode. This signal is also used for card detect in both 4-bit and 1-bit modes.
DAT2	This is a bidirectional signal for data transmission or read wait in 4-bit mode and is for read/wait when SDHC works in 1-bit mode
DAT1	This is a bidirectional signal for data transmission or interrupt signal in 4-bit mode and is for interrupt signal when SDHC works in 1-bit mode.
DAT0	This is a bidirectional signal for data transmission both in 1-bit and 4-bit mode.

34.3 Memory Map and Register Definition

SDHC contains 14 32-bit registers. This section provides the detailed descriptions for all SDHC registers. All registers must be accessed as 32-bit values. Byte/Half Word access is not allowed.

34.3.1 Memory Map

Table 34-3 shows the SDHC memory map. The SDHC memory map space is 4K. Only address offsets from 0x00 to 0x44 are implemented. The address space above offset of 0x44 is reserved. Write accesses to the reserved region are ignored. Read accesses to reserved locations return the value of 0x0000. Do not access the reserved region to ensure compatibility with possible future revisions of this module.

Table 34-3. SDHC Memory Map

Address	Register	Access
Base Address	SDHC Clock Control register	R/W
Base Address+04	SDHC Status register	R/W
Base Address+08	SDHC Card Clock Rate register	R/W
Base Address+0C	SDHC Command Data Control register	R/W
Base Address+10	SDHC Response Time-out register	R/W
Base Address+14	SDHC Read Time-out register	R/W
Base Address+18	SDHC Block Length register	R/W
Base Address+1C	SDHC Number of Block register	R/W
Base Address+20	SDHC Revision Number register	Read-only
Base Address+24	SDHC Interrupt Control register	R/W
Base Address+28	SDHC Command Number register	R/W
Base Address+2C	SDHC Command Argument register	R/W
Base Address+34	SDHC Command Response FIFO Access register	Read-only
Base Address+38	SDHC Data Buffer Access register	R/W

Note: The addresses (Base Address+ 0x30, Base Address+ 0x3C over Base Address+ 0x44) are reserved.

34.3.2 Register Descriptions

Many SDHC registers have reserved bits. Reserved bits in all registers are read as 0 and writes to these bits are ignored. However, software should write zeroes to these bits to ensure compatibility with possible future revisions of this module.

34.3.2.1 SDHC Clock Control Register (STR_STP_CLK)

The SDHC Clock Control Register allows software to reset the whole module and to enable or disable the MMC_SD_CLK to card.

See [Figure 34-2](#) for illustration of valid bits in the SDHC Clock Control Register and [Table 34-4](#) for description of the bit fields.

Offset 0x00Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													SLFC LR			
W													SDHC RESE T		START _CLK	STOP _CLK
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 34-2. STR_STP_CLK Register
(Register repeats for reference.)

Table 34-4. STR_STP_CLK Field Descriptions (Sheet 1 of 2)

Field	Description
SDHC RESET	SDHC Reset. Writes to the SDHC reset bit triggers the reset logic inside the SDHC. Reads from this bit always return 0. To reduce power consumption, the clock to the reset logic in SDHC is off in normal operation. When there is one access to this register, the clock is enabled for one cycle. To complete the entire reset period, it needs at least 8 clock pulses to finish the reset cycle. To reset the SDHC module, it is recommended to write this register with value 0x0008, followed by 0x0009, and then 0x0001 eight times. Refer to Section 34.5.3.2, “Reset” for detailed information on software reset. 0 No effect 1 Reset the SDHC module
START_CLK	Start Clock. Writing a 1 to this bit starts the MMC_SD_CLK clock. Setting a value of 11 on Bits [1:0] of this register is not allowed. Note: The SDHC bus clock does not start immediately after writing to this bit. Poll the CARD_BUS_CLK_RUN bit of the SDHC Status Register to ensure the SDHC clock is running. 0 Has no effect 1 To start MMC/SD clock

Offset 0x00Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													SLFC LR			
W													SDHC RESET		START_CLK	STOP_CLK
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 34-2. STR_STP_CLK Register
(Register repeats for reference.)

Table 34-4. STR_STP_CLK Field Descriptions (Sheet 2 of 2)

Field	Description
STOP_CLK	<p>Stop Clock. Stops the MMC_SD_CLK clock when software writes a value of 1 to this bit. Software should not stop the MMC_SD_CLK during a transmission period.</p> <p>Writing a value of 11 to bits [1:0] of this register is not allowed.</p> <p>Note: A transmission period is the time from when a card data or access related command is submitted to the end of the access operation.</p> <p>Note: The SDHC bus clock does not stop immediately after writing to this bit. Polling needs to be done on the status CARD_BUS_CLK_RUN bit of the SDHC status register to ensure the SDHC clock is not running</p> <p>0 Has no effect</p> <p>1 To stop the MMC/SD clock</p>

Table 34-6 summarizes the ipg_clk and SDHC_CLK status under different operation mode and clock gating setting.

34.3.2.2 SDHC Status Register (STATUS)

The read-only SDHC Status Register provides the programmer with information about the status of SDHC operations, application FIFO status, error conditions, and interrupt status.

There are eight interrupt status bits.

Table 34-5. Interrupt Status Bits

Bit	Interrupt
31	Card insertion status bit
30	Card removal status bit
14	SDIO card interrupt status bit
13	End command and response status bit
12	Write operation done status bit
11	Read operation done status bit
7	Data buffer read ready status bit
6	Data buffer write ready status bit.

When the corresponding interrupt enable is enabled in SDHC interrupt control register for any of these interrupts, SDHC generates an interrupt request to the CPU. User needs to clear the appropriate status bit to clear the corresponding interrupt. The interrupt status bits are cleared by using a write 1 to clear operation except for the data buffer ready status bits which can only be cleared by the read or write operation on the data buffer.

See [Figure 34-3](#) for illustration of valid bits in the SDHC Status Register and [Table 34-6](#) for description of the bit fields.

Offset 0x04 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CARD_INSERTION	CARD_REMOVAL	YBUF_EMPTY	XBUF_EMPTY	YBUF_FULL	XBUF_FULL	BUF_UNDRUN	BUF_OVFL								
W	w1c	w1c														
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		SDIO_INT_ACTIVE	END_CMD_RESP	WRITE_OP_DONE	READ_OP_DONE	WR_CRC_ERR_CODE		CARD_BUS_CLK_RUN	BUF_READ_RDY	BUF_WR_RDY	RESP_CRC_ERR		READ_CRC_ERR	WRITE_CRC_ERR	TIME_OUT_RESP	TIME_OUT_READ
W		w1c	w1c	w1c	w1c	w1c	w1c				w1c		w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 34-3. STATUS Register

Table 34-6. STATUS Field Descriptions

Field	Description
CARD_INSERTION	Card Insertion. When this bit is set, the SDHC detects a value transition on the DAT[3:0] from b0111 to b1111. This can detect whether a card is inserted to the card socket based on DAT3 pull-up resistor of the card DAT3. When this bit is set, the SDHC generates an interrupt request if the card detection interrupt is enabled. This bit is read only and can be cleared by writing 1 to this bit. 0 No card insertion detected 1 Card insertion detected based on logic level changed on DAT3
CARD_REMOVAL	Card Removal. When this bit is set, the SDHC detects a logic transition on the DAT[3:0] from b1111 to b0111. This can detect whether a card is removed from the card socket based on the DAT3 pull-up resistor of the card DAT3. When this bit is set, SDHC generates an interrupt request if the card removal interrupt is enabled. This bit is read only and can be cleared by writing a 1 to it. User needs to clear this bit to clear the interrupt request from SDHC when card detection interrupt is enabled. 0 No card insertion detected 1 Card removal detected based on logic level changed on DAT3
YBUF_EMPTY	Y Data Buffer Empty. When this is set, it indicates the Y data buffer is empty during a write transfer. This bit is automatically cleared when the first byte of data is moved into the FIFO. Refer to Section 34.4.1, “Data Buffers” for more information about the data buffers. 0 Y buffer is not empty 1 Y buffer is empty
XBUF_EMPTY	X Data Buffer Empty. When this is set, it indicates the X data buffer is empty during a write transfer. This bit is automatically cleared when the first byte of data is moved into the FIFO. Refer to Section 34.4.1, “Data Buffers” for more information about the data buffers. 0 X buffer is not empty 1 X buffer is empty
YBUF_FULL	Y Data Buffer Full. When this is set, it indicates the Y data buffer is full during a read transfer. This bit is automatically cleared when the last byte of data is read out from the FIFO. Refer to Section 34.4.1, “Data Buffers” for more information about the data buffers. 0 Y buffer is not full 1 Y buffer is full
XBUF_FULL	X Data Buffer Full. When set, this bit indicates the X data buffer is full during a read transfer. This bit is automatically cleared when the last byte of data is read out from the FIFO. Refer to Section 34.4.1, “Data Buffers” for more information about the data buffers. 0 X buffer is not full 1 X buffer is full
BUF_UND_RUN	Buffer Underrun. When set, this bit indicates both X and Y data buffers are empty during a write transfer. In this case, the card clock MMC_SD_CLK is stopped automatically by hardware to wait for the DMA or CPU to put data into the buffers. An interrupt is triggered if the corresponding interrupt control bit is enabled. 0 No buffer underrun 1 Buffer underrun during a write operation
BUF_OVFL	Buffer Overflow. When set, this bit indicates both data buffers are full during a read operation. In this case, the card clock MMC_SD_CLK is stopped automatically by hardware to wait for the DMA or CPU to remove data out of one of the buffers. An interrupt is triggered if the corresponding interrupt control bit is enabled. Excess data is ignored by the SDHC. 0 No buffer overflow 1 Buffer overflow during a read operation

Table 34-6. STATUS Field Descriptions (continued)

Field	Description
SDIO_INT_ACTIVE	<p>SDIO Interrupt Active. This indicates whether an interrupt from the SDIO card has been detected. When this bit is set, the SDHC generates an interrupt request if the SDIO interrupt is enabled. User should clear the status bit to clear the interrupt request. However, a separate acknowledge command to the card may be required to clear the source of the SDIO interrupt. Writing a 1 to this bit clears it.</p> <p>0 No interrupt detected 1 Interrupt detected using SDIO card bus</p>
END_CMD_RESP	<p>End Command Response. This indicates a command was successfully transmitted to the card and the corresponding response is stored in the Response FIFO. This occurs after each command operation. When this bit is set, SDHC generates an interrupt request if END_CMD_RESP interrupt is enabled. User needs to clear this bit to clear the interrupt request. Writing a 1 to this bit clears it.</p> <p>0 Command not successful, incomplete, or not applicable (no response) 1 Command transmitted successfully (response received)</p> <p>Note: When this bit is set, check the response stored in the response FIFO completed without fail. Also, check the RESP_CRC_ERR (Status[5]) and TIME_OUT_RESP(STATUS[1]) bits to determine if an error occurred.</p>
WRITE_OP_DONE	<p>Write Operation Done. This indicates a write operation has completed. The flash card might need extra idle time for write accesses, which requires the SDHC module to wait until the card writes the buffered data to the inner flash memory. WRITE_OP_DONE flag indicates the end of the write operation. When this bit is set, the pre-defined data bytes are written to the card. User needs to send a STOP command to the card if the write command is a MMC/SD card write multi-block command. When this bit is set, SDHC generates an interrupt request if the WRITE_OP_DONE interrupt enable is enabled. User needs to clear this bit to clear the interrupt. This is accomplished by writing 1 to this bit.</p> <p>0 Write operation in progress or incomplete 1 Write operation complete</p> <p>Note: When this bit is set, user also needs to check if the write operation completed without a cyclic redundancy check (CRC) error. Also, user needs to check the WR_CRC_ERR_CODE[1:0] (Status[10:9]) and WRITE_CRC_ERR (STATUS[2]) bits to determine if an error has occurred.</p>
READ_OP_DONE	<p>Read Operation Done. The READ_OP_DONE status bit is activated at the end of a read operation. When this bit is set, pre-defined data bytes have been read from the card or a READ TIMEOUT has occurred. Software needs to send a STOP command to card if the read command is a MMC or SD card read multi-block command. This bit can be cleared by writing 1 to it. When this bit is set, SDHC generates an interrupt request if the READ_OP_DONE interrupt is enabled. User needs to clear this bit to clear the interrupt request.</p> <p>0 Read operation in progress or incomplete 1 Read operation complete</p> <p>Note: When this bit is set, user also needs to check if the read operation complete without error. Also, user needs to check the READ_CRC_ERR (Status[3]) and TIME_OUT_READ(STATUS[0]) bits to determine if an error has occurred.</p>
WR_CRC_ERROR_CODE	<p>Write CRC Error Code. This indicates CRC results from the card at the end of write operations. After receiving a block of data, the card checks the CRC bit and sends the CRC status. These two bits reflect the CRC status of the recent written data. If card feedbacks a negative CRC status, data is not written to the card. These two bits can be cleared by writing a value of b11 to them.</p> <p>00 No transmission error, CRC status is 010 (positive) 01 Transmission error, CRC status is 110 (negative) 10 No CRC response 11 Reserved</p> <p>Note: The bits only have valid value when the WRITE_CRC_ERR status bit (STATUS[2]) is set.</p>

Table 34-6. STATUS Field Descriptions (continued)

Field	Description
CARD_BUS_CLK_RUN	Card Bus Clock Run. This indicates whether the MMC_SD_CLK clock to the card is running. The clock rate setting and system configuration can be modified when the clock is turned off by setting the STOP_CLK bit in STR_STP_CLK Register. This bit can only be cleared by writing 1 to STOP_CLK bit in STR_STP_CLK clock control register to stop MMC_SD_CLK. 0 MMC/SD clock is stopped 1 MMC/SD clock is running Polling needs to be done on this bit to assure the SDHC clock is running or stopped.
BUF_READ_RDY	Buffer Read Ready. This status is set if a buffer (either X buffer or Y buffer) is full during read operations. An interrupt is triggered for non-DMA transfers if BUF_READ_EN is set, or a DMA request is asserted for DMA transfers. 0 Not ready to read buffer 1 Ready to read buffer
BUF_WR_RDY	Buffer Write Ready. This status is set if a buffer (either X buffer or Y buffer) is available during write operations. An interrupt is triggered for non-DMA transfers if the BUF_WRITE_EN bit is set or a DMA request is asserted for DMA transfers. This bit is only set when SDHC is perform write operation to the card. 0 Not ready to write buffer 1 Ready to write buffer
RESP_CRC_ERR	Response CRC Error. This indicates a transmission error occurred on the SD_CMD line during a response transfer. Writing 1 to this bit clears this bit. 0 No error 1 Response CRC error occurred
READ_CRC_ERR	Read CRC Error. This indicates a transmission error occurred on the DAT line during a card read. User should retry the transmission. Writing a 1 to this bit clears the bit. 0 No error 1 CRC read error occurred
WRITE_CRC_ERR	Write CRC Error. This indicates a transmission error occurred on the DAT line during a card write operation. User should check the WR_CRC_ERR_CODE field for more information about the CRC error. Writing a 1 to this bit clears this bit. 0 No error 1 CRC write error occurred
TIME_OUT_RESP	Time Out Response. This indicates command response was not received in time specified in the RES_TO Register. This can be caused by: <ul style="list-style-type: none"> • An unsupported command received at the card(s). • Another MMC/SD_OP_COND command submitted after all cards had already sent their voltage ranges and the power-up routine is finished. • An identification command issued when all cards are already in standby state. • No card is on the bus. Writing a 1 to this bit clears this condition. 0 No error 1 Time out response error occurred
TIME_OUT_READ	Time Out Read. Indicates expected data from the card was not received in time specified in the READ_TO Register. The TIME_OUT_READ is cleared by an internal status change or by removing the source of the error. Writing a 1 to this bit clears this bit. 0 No error 1 Time out read data error occurred

34.3.2.3 SDHC Clock Rate Register (CLK_RATE)

Refer to [Section 34.4.7, “System Clock Controller,”](#) for the clock scheme.

The high frequency input clock, SDHC_CLK, derives the low frequency clock (CLK_20M) the card uses. The divide circuitry consists of a 4-bit divider followed by a 12-bit prescaler. The SDHC_CLK is first divided by the 4-bit divider to derive the signal, CLK_DIV. The 12-bit prescaler divides the CLK_DIV to derive CLK_20M, which the card can use.

CLK_20M is used internally by the SDHC.

See [Figure 34-4](#) for illustration of valid bits in the SDHC Clock Rate Register and [Table 34-7](#) for description of the bit fields.

Offset 0x08 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CLK_PRESCALER												CLK_DIVIDER			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 34-4. CLK_RATE Register
(Register repeats for reference.)

Table 34-7. CLK_RATE Field Descriptions

Field	Description
CLK_PRESCALER	Clock Prescaler. Specifies divider value to generate CLK_20M from CLK_DIV. 0x000 CLK_20M is CLK_DIV 0x001 CLK_20M is CLK_DIV/2 0x002 CLK_20M is CLK_DIV/4 0x004 CLK_20M is CLK_DIV/8 0x008 CLK_20M is CLK_DIV/16 0x010 CLK_20M is CLK_DIV/32 0x020 CLK_20M is CLK_DIV/64 0x040 CLK_20M is CLK_DIV/128 0x080 CLK_20M is CLK_DIV/256 0x100 CLK_20M is CLK_DIV/512 0x200 CLK_20M is CLK_DIV/1024 0x400 CLK_20M is CLK_DIV/2048 0x800 CLK_20M is CLK_DIV/4096

Offset 0x08 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CLK_PRESCALER												CLK_DIVIDER			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 34-4. CLK_RATE Register
(Register repeats for reference.)

Table 34-7. CLK_RATE Field Descriptions (continued)

Field	Description
CLK_DIVIDER	<p>Clock Divider. Specifies the divider value to generate CLK_DIV from input clock SDHC_CLK.</p> <p>0x0 CLK_DIV is SDHC_CLK</p> <p>0x1 CLK_DIV is SDHC_CLK/2</p> <p>0x2 CLK_DIV is SDHC_CLK/3</p> <p>0x3 CLK_DIV is SDHC_CLK /4</p> <p>0x4 CLK_DIV is SDHC_CLK /5</p> <p>0x5 CLK_DIV is SDHC_CLK /6</p> <p>0x6 CLK_DIV is SDHC_CLK /7</p> <p>0x7 CLK_DIV is SDHC_CLK /8</p> <p>0x8 CLK_DIV is SDHC_CLK /9</p> <p>0x9 CLK_DIV is SDHC_CLK /10</p> <p>0xa CLK_DIV is SDHC_CLK /11</p> <p>0xb CLK_DIV is SDHC_CLK /12</p> <p>0xc CLK_DIV is SDHC_CLK /13</p> <p>0xd CLK_DIV is SDHC_CLK /14</p> <p>0xe CLK_DIV is SDHC_CLK /15</p> <p>0xf CLK_DIV is SDHC_CLK /16</p> <p>Others Reserved</p>

NOTE

Maximum frequency of MMC_SD_CLK is SDHC_CLK/1 when CLK_DIVIDER and CLK_PRESCALAR are set to 0x0.

34.3.2.4 SDHC Command and Data Control Register (CMD_DAT_CONT)

SDHC Command and Data Control Register allows user to specify the format of data and response and control the Read/Wait cycle. After configuring this register, enabling the MMC_SD_CLK causes the command and argument configured in the CMD Number register and the CMD Argument register to be sent out to the card.

See [Figure 34-5](#) for illustration of valid bits in the SDHC Command and Data Control Register and [Table 34-8](#) for description of the bit fields.

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	CMD_RESUME			CMD_RESP_LONG_OFF	STOP_READ_WAIT	START_READ_WAIT	BUS_WIDTH	INIT			WRITE_READ	DATA_ENABLE	FORMAT_OF_RESPONSE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-5. CMD_DAT_CONT Register
(Register repeats for reference.)

Table 34-8. CMD_DAT_CONT Field Descriptions

Field	Description
CMD_RESUME	Command Resume. Restores Command and Data Control Register after READ/WAIT cycle for SDIO card. 0 Issues command to card 1 Does not issue command to card
CMD_RESP_LONG_OFF	Command Response Long Off. Allows STATUS[13] END_CMD_RESP bit to be self-cleared when condition to generate this bit disappears. This is used in the Read/Wait cycle. For SD/MMC operation, keep this bit at 0. 0 Bit not cleared when read 1 Allows bit clearance
STOP_READWAIT	Stop Read/Wait. Ends the Read/Wait cycle for SDIO. When this bit is set, SDHC does not drive DAT2 output low so the SDIO card would end the Read/Wait cycle. For operation of SD/MMC, keep this bit at 0. 0 No effect 1 Ends Read/Wait cycle
START_READWAIT	Start Read/Wait. Starts the Read/Wait cycle for SDIO. When this bit is set, SDHC makes the DAT2 output low and forces the SDIO card to enter READWAIT cycle. For SD/MMC operation, keep this bit at 0. 0 No Effect 1 Starts Read/Wait cycle
BUS_WIDTH	Bus Width. Specifies the width of the data bus. These two bits must be set according to current card bit mode. 00 1-bit 01 Reserved 10 4-bit 11 Reserved

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	CMD_RESUME			CMD_RESP_LONG_OFF	STOP_READ_WAIT	START_READ_WAIT	BUS_WIDTH	INIT			WRITE_READ	DATA_ENABLE	FORMAT_OF_RESPONSE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-5. CMD_DAT_CONT Register
(Register repeats for reference.)

Table 34-8. CMD_DAT_CONT Field Descriptions (continued)

Field	Description
INIT	Initialize. Specifies whether the additional 80-clock (MMC_SD_CLK) cycle prefix (to initialize the card) occurs before every command. INIT enables/disables the additional 80-clock initialization time. 0 Disable 80 initialization clocks 1 Enable 80 initialization clocks
WRITE_READ	Write/Read. Specifies whether the data transfer of current command is a write or read operation 0 Read 1 Write
DATA_ENABLE	Data Enable. Specifies whether the current command includes a data transfer. 0 No data transfer included 1 Data transfer include
FORMAT_OF_RESPONSE	Format of Response. Sets the expected response format for current command. Refer to the SD I/O Specification 1.0 for detail information of the response format. 000 No response for Current command 001 Format R1/R5/R6 (48-bit Response with CRC7) 010 Format R2 (136-bit, CSD/CID read response) 011 Format R3/R4(48-bit Response without CRC check) OthersReserved

34.3.2.5 SDHC Response Time Out Register (RES_TO)

The MMC/SD Response Time Out Register defines an interval within which a response must be returned or a timeout error occurs. After SDHC sends out a command, if the card does not respond within the specified interval, the RESPONSE TIMEOUT status bit (STATUS[1]) and the END_CMD_RESP status bit (STATUS[13]) are set.

See [Figure 34-6](#) for illustration of valid bits in the MMC/SD Response Time Out Register and [Table 34-9](#) for description of the bit fields.

Offset 0x10Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									RESPONSE TIME OUT							
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-6. SDHC RES_TO Register

Table 34-9. RES_TO Field Descriptions

Field	Description
RESPONSE TIME OUT	Response Timeout. This value determines the interval which detects response timeout. The clock starts counting when the last bit of the command is sent. The clock counts unit is MMC_SD_CLK to card. 0x01 1 clock count 0x02 2 clock counts 0xFF 255 clock counts

34.3.2.6 SDHC Read Time Out Register (READ_TO)

The MMC/SD Read Time Out Register defines an interval that read data must be returned within or a timeout error occurs. After SDHC sends out data read command, if no data is returned within the specified interval, the READ TIMEOUT status bit (STATUS[0]) and the READ_OP_DONE status bit (STATUS[11]) are set.

See [Figure 34-7](#) for illustration of valid bits in the SDHC Read Time Out Register and [Table 34-10](#) for description of the bit fields.

Offset 0x14 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA_READ_TIME_OUT															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 34-7. READ_TO Register

Table 34-10. READ_TO Field Descriptions

Field	Description
DATA_READ_TIME_OUT	<p>Data Read Timeout. This value determines the interval read data timeouts are detected. Check the timeout limit of the card and clock frequency to configure this register. For safety, 0xFFFF is recommended.</p> <p>The time-out clock starts counting when the last bit of the command is sent. The count unit is MMC_SD_CLK/256. The maximum delay SDHC can tolerate for a data time out relates to the card clock. If the clock is 25 MHz and register is 0xFFFF, the maximum delay SDHC waits is about 670ms. If the card does not give data in 670ms, SDHC issues a data read time out error and terminates current data read operation. This meets the SD physical layer specification, with typical time out limit of 100ms~200ms. However, for some SDIO cards, the time out limit may be up to 1s. In such cases, lower the MMC_SD_CLK frequency to accommodate the delay to 1s, which user needs to configure the MMC_SD_CLK to about 16 MHz and set this register to be 0xFFFF.</p>

34.3.2.7 SDHC Block Length Register (BLK_LEN)

SDHC Block Length Register defines the number of bytes in a block. Because stream mode of MMC is not supported, block length must be set for every transfer. Block length supported by the SDHC ranges from 1 to 2048 bytes, but you need to check the block size supported by the card before configuring this register. For SDIO, block length must be less than the maximum block size defined in the card's CCCR. For SD/MMC, block length must be less than the maximum block size defined in the card's CSD register.

NOTE

Software should write to this register only when no SD bus transaction is executing.

See [Figure 34-8](#) for illustration of valid bits in the SDHC Block Length Register and [Table 34-11](#) for description of the bit fields.

Offset 0x18 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					BLOCK LENGTH											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-8. BLK_LEN Register

Table 34-11. BLK_LEN Field Descriptions

Field	Description
BLOCK LENGTH	<p>Block Length. Specifies the number of bytes in a block during data transfer (block size). For MMC and SD cards, the value set must remain the same as the Balkan set in the CARD. For SDIO, IO access is performed through the CMD53 IO_RW_EXTEND command. Command has two modes:</p> <ul style="list-style-type: none"> • Byte mode. For byte mode, its operation is similar to a single block transfer command for SD where the block length is the byte count in the command argument. • Block mode. For block mode, its operation is similar to a multi-block transfer command for the SD where the block length is the block size defined in the command argument. <p>For multi-block data transfers, a block length equal to an integer multiple of the data buffer size is preferred. Otherwise, buffer utilization is poor. If the data size that needs to be transferred is not an integer multiple of the buffer size, there are two options to transfer the data:</p> <ul style="list-style-type: none"> • Option 1: Split the transaction. The remainder of block size data is transferred by using a single block command for the last transfer. • Option 2: Add filler data in the last block to fill the block size to be as large as the buffer size. <p>The data buffer size is 64 bytes in 4-bit mode and 16 bytes in 1-bit mode. Refer to Section 34.4.1, “Data Buffers,” for more information about data buffer.</p> <p>0x000 0 byte 0x001 1 byte 0x7FF 2047 bytes 0x800 2048 bytes 0x801~0XFFF Reserved</p>

34.3.2.8 SDHC Number of Blocks Register (NOB)

The SDHC Number of Blocks Register defines the number of blocks in the block transfer mode. This register and the Block Length Register determines the number of bytes to be transferred during one command. The number decrements every time a block transfer completes and stops when the count reaches zero. When all data transfers are completed, the STATUS[11] READ_OP_DONE is set if it is a read (from card) transfer, or the STATUS[12] WRITE_OP_DONE is set if it is a write (to card) transfer.

Software should write to this register only when no MMC/SD transaction is executing.

Max data size to be transferred in bytes equals block length multiplied by number of blocks.

See [Figure 34-9](#) for illustration of valid bits in the SDHC Number of Blocks Register and [Table 34-12](#) for description of the bit fields.

Offset 0x1CAccess: User read/write

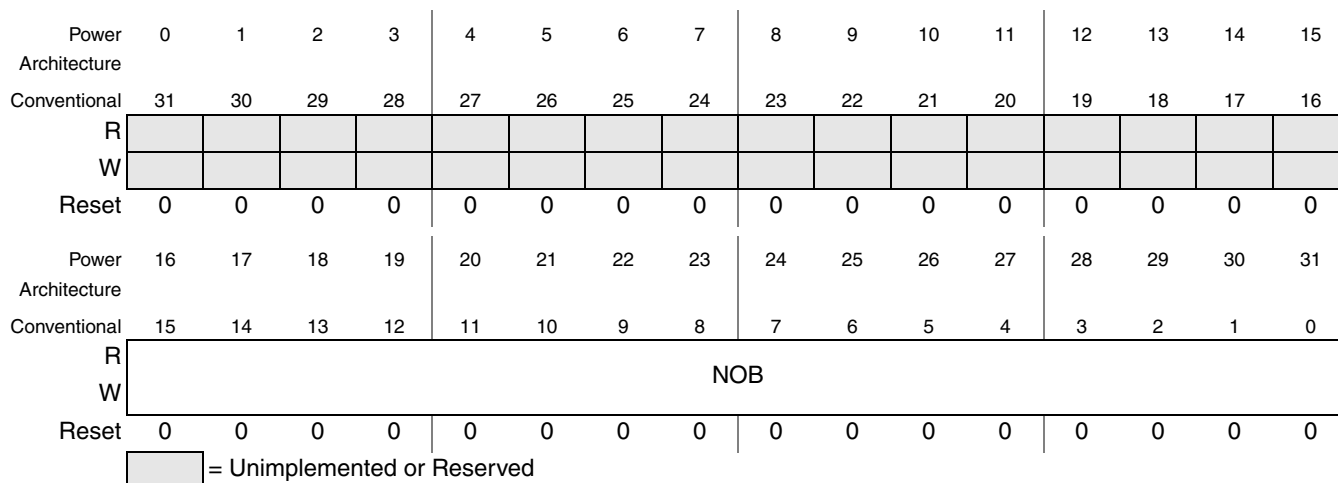


Figure 34-9. NOB Register

Table 34-12. NOB Field Descriptions

Field	Description
NOB	<p>Specifies the number of blocks in a block transfer. One block should be set if the data transfer command is a single block transfer command or IO_RW_EXTEND (CMD53) in byte mode. For multi-block transfer command to SD/MMC card and IO_RW_EXTEND (CMD53) in block mode to SDIO card, this register should be set the block count software expects. Number of blocks can range from 0 to 65535.</p> <p>For SD Memory card or a memory parts of a SDIO combo card, send CMD12 to stop the multi-block transfer. For a SDIO CMD53 in block mode and user needs to abort the transfer earlier, use CMD52 IO-Absort to abort the transfer.</p> <p>0x0000 0 Block 0x0001 1 Block 0xFFFF 65535 Blocks</p> <p>Note: The maximum transfer blocks is 64 K. If user uses infinite transfer command to transfer data, such as multi-block transfer command for memory card or infinite block transfer CMD53 for SDIO card, this register needs to be set to the real number of blocks that you expect to transfer. Also, you need to abort the transfer using CMD12 or CMD52 IO-Absort.</p>

34.3.2.9 SDHC Revision Number Register (REV_NO)

The SDHC Revision Number Register is a read-only register displaying the revision number of the module.

See [Figure 34-10](#) for illustration of valid bits in the SDHC Revision Number Register and [Table 34-13](#) for description of the bit fields.

Offset 0x20 Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Revision Number[15:0]															
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-10. REV_NO Register

Table 34-13. REV_NO Field Descriptions

Field	Description
REVISION NUMBER	Revision Number. Specifies revision number of the MMC/SD module. This is fixed at 0x0000_0400.

34.3.2.10 SDHC Interrupt Control Register (INT_CNTR)

When certain events occur in the module, the SDHC has the ability to set an interrupt as well as set corresponding status register bits. The SDHC Interrupt Control Register allows control over whether these interrupts should be recognized. Interrupts are ORed to provide a single interrupt IPI_IRQ to the system. Software must read the status to determine the source of the event.

See [Figure 34-11](#) for illustration of valid bits in the SDHC Interrupt Control Register and [Table 34-14](#) for description of the bit fields.

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CARD_INSERTION_EN	CARD_REMOVAL_EN	SDIO_IRQ_EN	DAT0_EN									BUF_READ_EN	BUF_WRITE_EN	END_CMD_RES	WRITE_OP_DONE
W																READ_OP_DONE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-11. INT_CNTR Register
(Register repeats for reference.)

Table 34-14. INT_CNTR Field Descriptions (Sheet 1 of 2)

Field	Description
CARD_INSERTION_EN	<p>Card Insertion Enable. Setting this bit enables the card insertion interrupt. Because card detection is through the value of DAT3 data line, if this card is in 4-bit mode, any data transfers in the DAT3 line causes false card insertion interrupts to be generated. Card insertion interrupt should be disabled after the first time card insertion is detected. To avoid false status bit generation during data transfer, card insertion status is masked by this bit. It should only be enabled after the card is removed from the socket.</p> <p>The default of this bit is to disable the card insertion interrupt. When this interrupt is detected, write a 1 to the STATUS[31] bit to clear the card insertion status interrupt.</p> <p>0 Card insertion interrupt disabled 1 Card insertion interrupt enabled</p>
CARD_REMOVAL_EN	<p>Card Removal Enable. Setting this bit enables the card removal interrupt. Because card detection is through the value of the DAT3 data line, if this card is in 4-bit mode, the data transfer through the DAT3 line causes false card removal interrupt to be generated. The card removal interrupt should only be enabled when no active data transfers exist on the DAT3 line. To avoid the false status bit generation during data transfer, the card insertion status is masked by this bit.</p> <p>The default of this bit is to disable the card removal interrupt. When this interrupt is detected, write a 1 to the STATUS[30] bit to clear the card removal status interrupt.</p> <p>0 Card removal interrupt disabled 1 Card removal interrupt enabled</p> <p>Note: SDHC uses IPS_CLK to detect the SDIO card interrupt wakeup event when this bit is set.</p>
SDIO_IRQ_EN	<p>SDIO Interrupt Enable. Masks the interrupt from the SD I/O card to the SDHC module interrupt.</p> <p>0 SD I/O interrupt disabled 1 SD I/O interrupt enabled</p> <p>Note: SDHC uses IPS_CLK to detect the SDIO card interrupt wakeup event when this bit is set.</p>

Offset 0x24 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CARD_INSERTION_EN	CARD_REMOVAL_EN	SDIO_IRQ_EN	DAT0_EN									BUF_READ_EN	BUF_WRITE_EN	END_CMD_RES	WRITE_OP_DONE
W																READ_OP_DONE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-11. INT_CNTR Register
(Register repeats for reference.)

Table 34-14. INT_CNTR Field Descriptions (Sheet 2 of 2)

Field	Description
DAT0_EN	Data Enable. Identifies how the SD I/O interrupt is detected. An interrupt is determined by DAT 1 low, but this bit is an optional setting for the SDIO bit. When SDHC preforms data transfer and the SD bus mode is 1-bit mode, set this bit to 0. 0 SD I/O's Interrupt detection based on DAT[3:0] = b110x 1 SD I/O's Interrupt detection based on DAT[3:0] = b1101
BUF_READ_EN	Bus Read Enable. This bit controls the buffer read ready interrupt. If the bit is 1, the interrupt is enabled. When the buffer becomes full during a read operation, an interrupt is generated. Move the data out of the FIFO and clear the BUF_READ_RDY bit to clear the interrupt. 0 Buffer status interrupt disabled 1 Buffer status interrupt enabled
BUF_WRITE_EN	Bus Write Enable. This bit controls the buffer write ready interrupt. If the bit is 1, interrupt is enabled. When the buffer becomes empty during a write operation, an interrupt is generated. Write data to the FIFO and clear the BUF_WRITE_RDY bit to clear the interrupt. 0 Buffer status interrupt disabled 1 Buffer status interrupt enabled
END_CMD_RES	End Command Response. This bit controls the interrupt generation on the status at the end of the command response. When this bit is 1, SDHC generates an interrupt at the end of the command response status. 0 End Command-response interrupt disabled 1 End Command-response interrupt enabled
WRITE_OP_DONE	Write Operation Done. This bit controls the interrupt generation for the status of write operation. When the interrupt enabled, SDHC generates an interrupt when the configured bytes of data are transferred to the card. 0 Write_OP_DONE interrupt disabled 1 Write_OP_DONE interrupt enabled
READ_OP_DONE	Read Operation Done. This bit controls the interrupt generation for the status of read operation completion. When the interrupt is enabled, SDHC generates an interrupt when the pre-defined bytes of data are transferred from the card. 0 READ_OP_DONE interrupt disabled 1 READ_OP_DONE interrupt enabled

When an interrupt is generated, there may be some error bits in the STATUS register set and the pending interrupt status (source of the interrupt must be cleared). Check the error status bit to make sure there is no error in the SDHC operation. For example, when the READ_OP_DONE (STATUS[11]) status is set or the READ_OP_DONE interrupt is detected, check the STATUS[3] and STATUS[0] bits to make sure the read operation completed without a CRC error or a time out error. Another example is a write operation, if both the WRITE_OP_DONE(STATUS[12]) and WRITE_CRC_ERR (STATUS[2]) bits are set. This means the write operation ended with CRC error. See [Table 34-15](#) for a summary of the relationship between the interrupt, interrupt control register, and status registers in the SDHC.

Table 34-15. Interrupt Mechanisms

Source STATUS Bit Name (Status Bit Number)	Does status directly generate interrupt?	INT_Control Register Bit Name (INT_CNTR bit Number)	Interrupt/Status Clear Method
TIME_OUT_READ (0)	No, alert using the READ_OP_DONE bit in the SDHC Status Register.	READ_OP_DONE (0)	Clear status by writing 1
TIME_OUT_RESP (1)	No, alert using the END_CMD_RESP bit in the SDHC Status Register.	END_CMD_RES (2)	Clear status by writing 1
WRITE_CRC_ERR (2)	No, alert using the WRITE_OP_DONE bit in the SDHC Status Register.	WRITE_OP_DONE (1)	Clear status by writing 1
READ_CRC_ERR (3)	No, alert via the READ_OP_DONE bit in the SDHC Status Register.	READ_OP_DONE (0)	Clear status by writing 1
RESP_CRC_ERR (5)	No, alert using the END_CMD_RESP bit in the SDHC Status Register.	END_CMD_RES (2)	Clear status by writing 1
BUF_WR_RDY (6)	Yes	BUF_WRITE_EN (3)	Clear status by writing data to FIFO buffer
BUF_READ_RDY (7)	Yes	BUF_READ_EN (4)	Clear status by reading data from FIFO buffer
READ_OP_DONE(11)	Yes	READ_OP_DONE (0)	Clear status by writing 1
WRITE_OP_DONE(12)	Yes	WRITE_OP_DONE (1)	Clear status by writing 1
END_CMD_RESP(13)	Yes	END_CMD_RESP (2)	Clear status by writing 1
SDIO_INT_ACTIVE(14)	Yes	SDIO_IRQ_EN (13)	Clear status by writing 1
CARD_REMOVAL(30)	Yes	CARD_REMOVAL_EN (14)	Clear status by writing 1
CARD_INSERTION(31)	Yes	CARD_INSERTION_E N (15)	Clear status by writing 1

34.3.2.11 SDHC Command Number Register (CMD)

The command to the SD card is always 48 bits long. It contains 1 start bit, 1 direction bit, 6 command number bits, 32 argument bits, 7 CRC bits, and 1 end bit. For more details on the format of the command,

refer to the SD physical layer spec. Refer to related card spec for detailed information about each command.

SDHC automatically generates the command start bit, direction bit, CRC7 bits, and end bit in hardware. Configure only the SDHC command number register and SDHC command argument register to issue a command to the card.

See [Figure 34-12](#) for illustration of valid bits in the SDHC Command Number Register and [Table 34-16](#) for description of the bit fields.

Offset 0x28 Access: User read/write

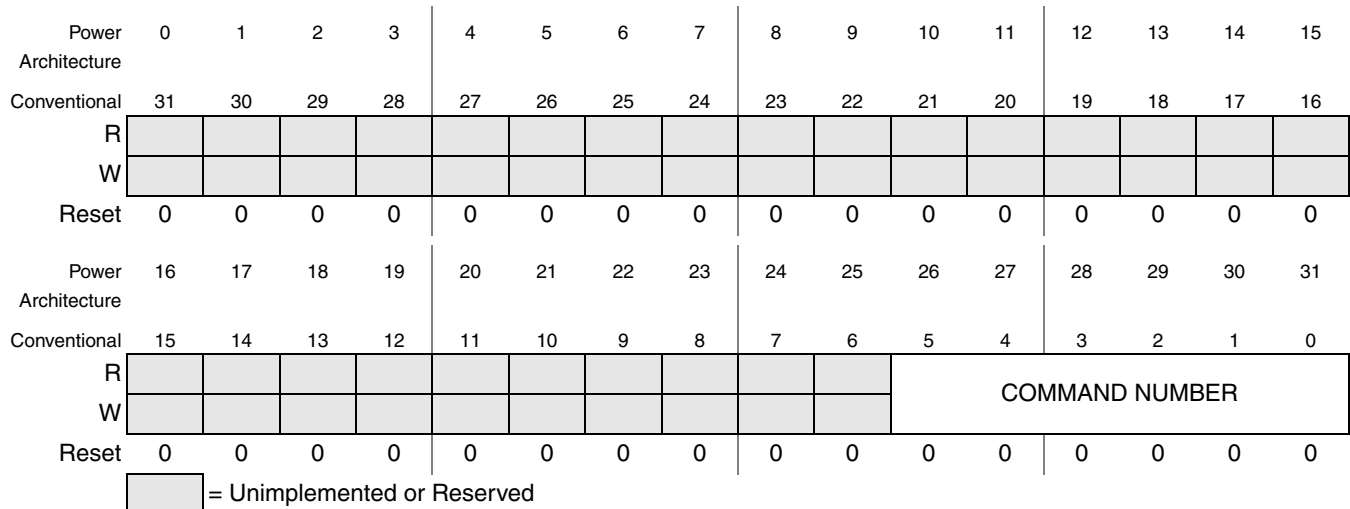


Figure 34-12. CMD Register

Table 34-16. CMD Field Descriptions

Field	Description
COMMAND NUMBER	<p>Command Number. The SDHC module communicates with the MMC/SD/SDIO card(s) by sending commands and arguments. The command to send is set in the MMC/SD Command Number Register (CMD), and the argument is defined in SDHC CMD Argument Register (ARG).</p> <p>0x00 CMD0 0x01 CMD1 0x3F CMD63 Note: Check the detail information from the related card spec.</p>

34.3.2.12 SDHC CMD Argument Register (ARG)

This register contains the MMC/SD/SDIO command argument.

See [Figure 34-13](#) for illustration of valid bits in the SDHC Command Argument Register and [Table 34-17](#) for description of the bit fields.

Offset 0x2CAccess: User read/write

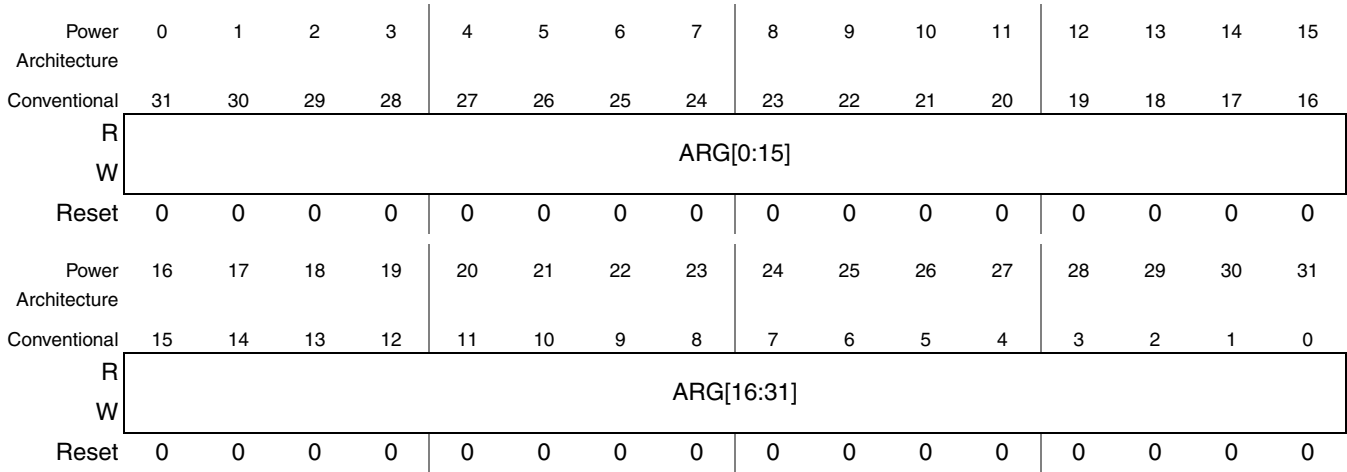


Figure 34-13. ARG Register

Table 34-17. ARG Field Descriptions

Field	Description
ARG	Command Argument. Specifies the argument for the current command. Note: Check the detail command argument information from the related card spec.

34.3.2.13 SDHC Response FIFO Access Register (RES_FIFO)

There is an 8 x 16 bit FIFO in the SDHC used to store the response from the card. The FIFO data can be read using this register. The most significant 16 bits of the response are accessed first, and the least significant 16 bits are accessed last.

See [Figure 34-14](#) for illustration of valid bits in the SDHC Response FIFO Access Register and [Table 34-18](#) for description of the bit fields.

Offset 0x34 Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RESPONSE_CONTENT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 34-14. RES_FIFO Register

Table 34-18. RES_FIFO Field Descriptions

Field	Description
RESPONSE_CONTENT	<p>RESPONSE CONTENT FIFO access register. There is a FIFO in the SDHC that stores the command response received from the card. Every time the Host sends a command to a card, the current contents stored in the FIFO are cleared and a new response argument is stored into the response FIFO.</p> <p>According to the SD card spec, command response size can be 48 bit or 136 bit (R2 response). Refer to the SD Memory Card Specification for more detailed information about the command response format. The response FIFO is 8x16 bits (128 bits). For a 48-bit response, only 48 bits of the FIFO have valid contents and user must perform three reads to this response FIFO access register to retrieve the entire 48-bit response content. For a 136-bit R2, response (response for CID[127:0] or CSD[127:0] register), only the contents of the 128-bit CID and CSD register are stored in the response FIFO. This first byte of the R2 response is not stored in the response FIFO. Retrieve the CIS/CSD register from the response FIFO through eight accesses to the FIFO access register. The CRC bit in the response is not stored in the response FIFO. This response FIFO is read only.</p>

34.3.2.14 SDHC Data Buffer Access Register (DBA)

The SDHC uses two 64-byte data buffers in an alternating manner to transfer data by the DMA and the SD card simultaneously to maximize throughput between the two clock domains (the IP peripheral clock, SDHC_CLK, and the host clock, CLK_20M). These buffers are temporary storage for data transferred between the host system and the card and vice versa. Read or write data to the buffers through this buffer access register. Refer to [Section 34.4.1, “Data Buffers,”](#) for more information about the data buffers.

In the read operation, SDHC stores data received from the card into the buffer. Move the data out of the buffer when the buffer is full.

In the write operation, SDHC fetches data from the buffer and transfers it to the card. Access the data buffer through the SDHC data buffer access register. Move data into the buffer when the buffer is empty.

See [Figure 34-15](#) for illustration of valid bits in the SDHC data buffer access register and [Table 34-19](#) for description of the bit fields.

Offset 0x38 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FIFO CONTENT[0:15]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FIFO CONTENT[16:31]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-15. BUFFER_ACCESS Register

Table 34-19. BUFFER_ACCESS Field Descriptions

Field	Description
FIFO CONTENT	FIFO Content. These bits hold 32-bit data upon a read or write transfer. The size of the FIFO is 4x32 bits (16 bytes in total) for SD 1-bit mode and 16x32 bits (64 bytes in total) for SD 4-bit mode. For reception, SDHC controller generates a DMA request when FIFO is full. Upon receiving this request, DMA starts transferring data from the SDHC FIFO to system memory by reading the data buffer access register for a number of pre-defined bytes. For transmit, SDHC controller generates a DMA request when FIFO is empty. Upon receiving this request, DMA starts moving data from the system memory to the SDHC FIFO by writing to the data buffer access register for a number of pre-defined bytes.

34.4 Functional Description

The Secure Digital Host Controller module (SDHC) controls the MMC, SD memory card, and I/O cards by sending commands to cards and performing data accesses to/from the cards.

The following sections provide a brief functional description of the major system blocks, including the DMA interface, memory controller, logic/command controller, and system clock controller.

34.4.1 Data Buffers

The SDHC uses two data buffers in an alternating manner to transfer data through the DMA and the SD card simultaneously to maximize throughput between the two clock domains (the IP peripheral clock, IPG_PERCLK, and the host clock, CLK_20M). See [Figure 34-16](#) for illustration of the buffering scheme. These buffers are used as temporary storage for data transferred between the host system and the card.

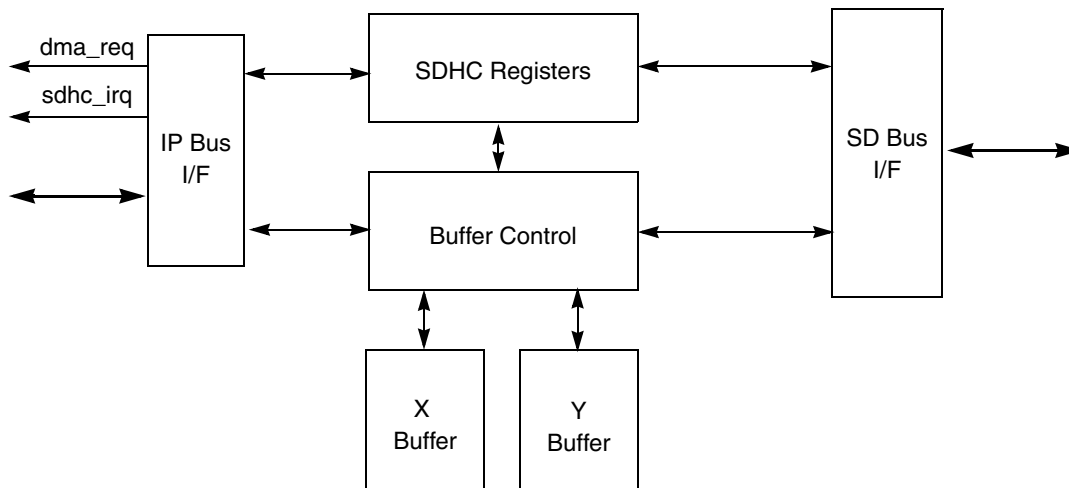


Figure 34-16. SDHC Buffer Scheme

For a host read operation, the SDHC automatically transfers data into the next available buffer. If one of the the data buffers is full, the SDHC generates a DMA request. Conversely, for a host write operation, if one of the data buffers is empty, the SDHC generates DMA requests. If some data is available, the SDHC reads the data out of the buffer and writes it to the card through the SD bus interface.

34.4.1.1 Data Buffer Access

The DMA/CPU accesses the SDHC data buffer as a FIFO through the 32-bit data buffer access (DBA) register. Internally, the SDHC maintains a pointer into the data buffer. Accesses to the DBA register automatically increase the pointer value. The pointer value is not directly accessible by the software. In cases when the block length of the data transfer is not a multiple of 32-bit, the last access to the DBA contains valid data only on 8, 16, or 24 bits. Because SDHC data buffer only allows 32-bit accesses, put/fetch the data bytes on the correct byte position of the DBA. For an 8-bit data access to the FIFO, put/fetch data into DBA bits 7 through 0. For 16-bit data access, put/fetch data in DBA bits 15 through 0. For a 24-bit data, put/fetch data into DBA bits 23 through 0.

When data is written to the card, a 32-bit data word in the data buffer is shifted out from the LSB byte to the MSB byte. When data is read from the card, the data is shifted to the data buffer from LSB byte to MSB byte. See [Figure 34-17](#) for the data swap between system IP bus and SDHC data buffer.

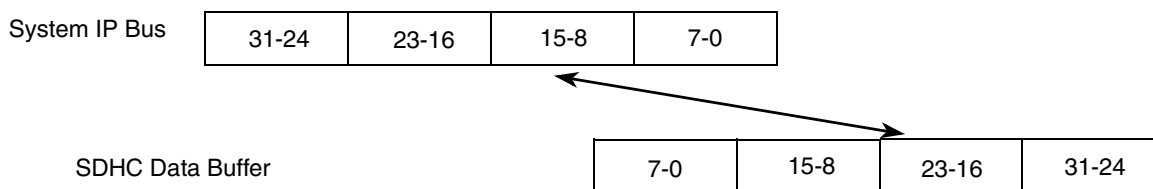


Figure 34-17. Data Swap Between System IP Bus and SDHC Data Buffer

34.4.1.2 Write Operation Sequence

There are two ways to move data into the SDHC data buffer when you want to write data to the card. One is by using DMA through the SDHC DMA request signal, and the other is by using the CPU through the BUF_WR_RDY (STATUS[6]) bit (interrupt or polling).

The SDHC automatically asserts a DMA request when the data buffer is empty and it is ready to receive new data. At the same time, SDHC sets the BUF_WR_RDY (STATUS[6]) bit. The buffer write ready interrupt is generated if enabled by software.

The data buffer accumulates the data written until it fills. The SDHC does not start writing to the card until a data buffer size is full. Then, the SDHC starts a transmission when the SD bus is ready for a new transfer. When the other data buffer is empty and more data can be transferred, SDHC asserts a new DMA request and sets the BUF_WR_RDY bit. See [Section 34.4.2.1, “DMA Request”](#).

34.4.1.3 Read Operation Sequence

There are two ways to fetch data from the buffer when data is read from the card. One is by using DMA through the SDHC DMA request signal, and the other is by using CPU through the BUF_READ_RDY (STATUS[7]) bit (interrupt or polling).

The SDHC asserts a DMA request when the data buffer is full and it is ready for DMA/CPU to fetch data out of the data buffer. At the same time, SDHC sets the BUF_READ_RDY (STATUS[7]) bit. The buffer read ready interrupt is generated if enabled by software.

SDHC only starts receiving data when either of the two data buffers is empty. The buffer accumulates data read from the card until it fills. SDHC asserts a DMA request when either one of the data buffers is full. For multiple block data transfers, while the DMA/CPU moves data by reading the DBA register, SDHC receives data into the other buffer if empty and the SD bus is ready. If the DMA/CPU does not keep up with reading data out of the buffers, SDHC stops the SD_CLK at the block gap to avoid an overflow situation.

34.4.1.4 Data Buffer Size

You must know the buffer sizes during data transfers. In SDHC, both data buffers are 64 bytes in size. However, each data buffer is divided into four 16 byte containers that correspond to the four data lines of SD bus. Therefore, the data buffer size is 64 bytes in 4-bit SD mode and 16 bytes in 1-bit SD mode.

During multi-block data transfer, block length should be an integer multiple of the buffer size. The buffer is ready to be read by CPU/DMA when either of the buffers is full (STATUS[27] or STATUS[26] is set, and STATUS[7] is set). The buffer would be ready to write by CPU/DMA (STATUS[29] or STATUS[28] is set, and STATUS[6] is set) when the full buffer of data are fetched out of the buffer. The buffer ready status bit and DMA request are set accordingly.

For single block data transfers when the block length is smaller than the buffer size or when the block length is not an integer multiple times that of the buffer size, the data size may need to be written to the buffer or to be fetched out of the buffer in smaller records. In this case, the buffer would be full (SDHC set STATUS[27] or STATUS[26]) when these data are written to the buffer. The buffer would be empty (SDHC set STATUS[29] or STATUS[28]) when these buffer of data are fetched out of the buffer. The

buffer ready status bit and DMA request would be set accordingly. From the software point of view, buffer size becomes variable and equal to the real data size that needs to be transferred. This eases the software programming of SDHC. Do not need to fill dummy data to make the buffer full.

34.4.1.5 Dividing Large Data Transfer

This SDIO command CMD53 definition limits the maximum size of data transfers according to this formula.

$$\text{Max Transfer Size} = \text{Block Size} \times \text{Block Count}$$

Eqn. 34-1

The block size can be a multiple of the size of the data buffer. However, it is recommended that the block size be equal to the data buffer size. This allows the SDHC to stop the SD_CLK during block gaps should an overflow or underrun condition occur. Stopping the SD_CLK while the data lines are active may cause data corruption on some cards. If an application or card driver needs to transfer larger sets of data, the host driver should divide the data set into multiple blocks.

The length of a multiple block transfer needs to be in block size units. If the total data length cannot be divided evenly to a multiple of the block size, there are two ways to transfer the data depending on function and card design. Option one is for the card driver to split the transaction into a block transfer to send most of the data and a byte transfer to send the remaining data. Option two is to add filler data in the last block to complete the block size. For option two, the card must manage to remove the filler data.

See [Figure 34-18](#) for an example showing dividing a large data set. The 544-byte WLAN frame is divided into eight 64-byte blocks plus the block. Eight 64-byte blocks are sent in block transfer mode and the remaining 32 bytes are sent in byte transfer mode.

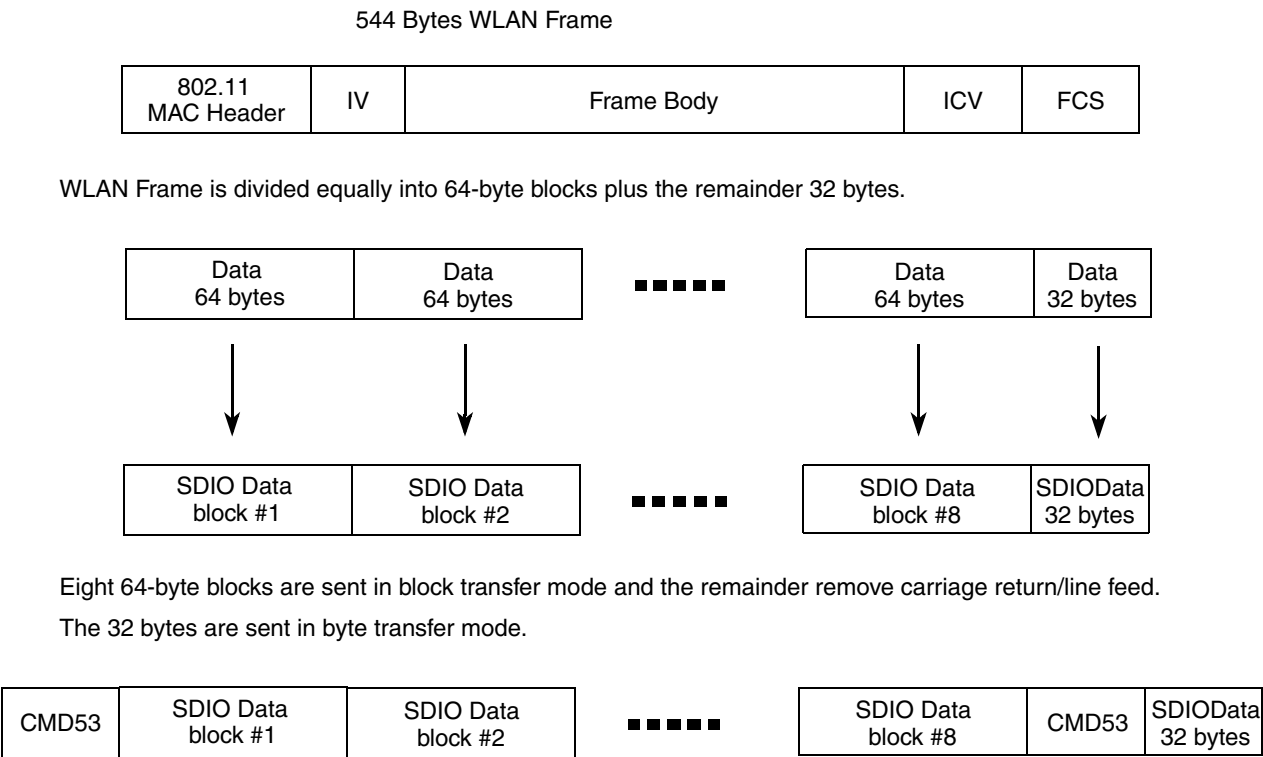


Figure 34-18. Example for Dividing Large Data Transfer

34.4.2 DMA Interface

The DMA interface block controls all data routing between the external data bus (DMA access), internal SDHC module data bus, and internal system FIFO access through a dedicated state machine that monitors the status of FIFO content (empty or full), FIFO address, and byte/block counters for the SDHC module and the application. See [Figure 34-19](#) for illustration of the DMA interface block.

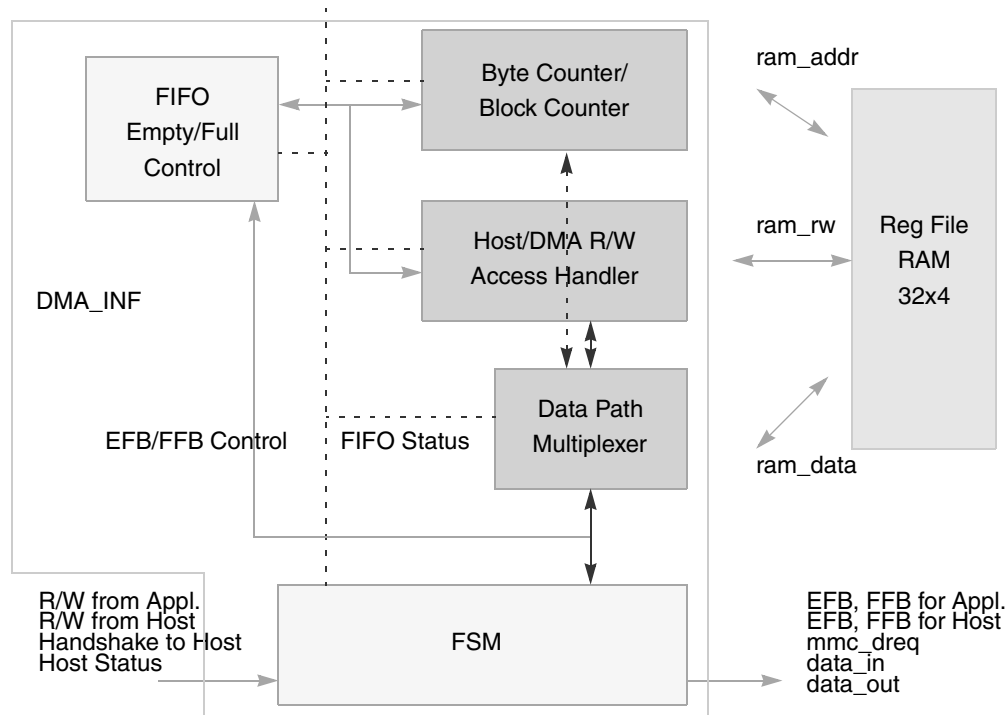


Figure 34-19. DMA Interface Block

In addition, this block also manages the burst request to the external DMA controller, internal register write-error detection, read wait handling of SDIO, and all IP related output responses.

34.4.2.1 DMA Request

If a transfer is in progress, SDHC generates DMA requests according to the FIFO status. During read operations, SDHC generates DMA requests if one of the data FIFOs is full. During write operations, SDHC generates DMA requests if one of the data FIFOs is empty.

To avoid buffer under-run conditions during a write operation, MMC_SD_CLK stops automatically when both buffers are empty. After the DMA or CPU completes writing data into one of the buffers, MMC_SD_CLK resumes automatically to continue the data transfer.

Similarly, to avoid buffer over-flow during read operations, MMC_SD_CLK stops automatically when both buffers are full. After the DMA or CPU moves the data out of the buffer, MMC_SD_CLK resumes automatically to continue the data transfer.

34.4.3 Memory Controller

This controller provides the SDIO-IRQ and read/wait service handling, card detection, command response handling, all SDHC interrupt managing, and it contains the register table. See [Figure 34-20](#) for illustration of the block diagram for the memory controller.

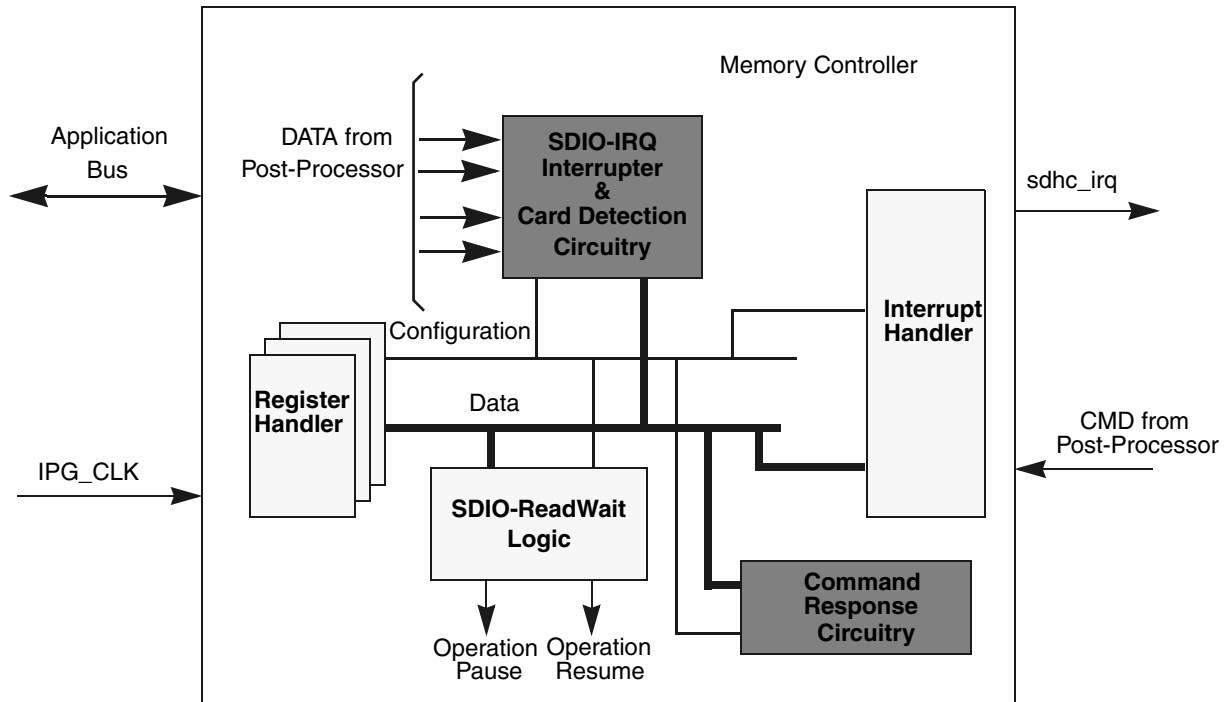


Figure 34-20. Memory Controller Block Diagram

A summary of events that take place when a SDIO card generates an interrupt is detailed in this section. When a SDIO card generates an interrupt request, it sets its interrupt pending bit in the CSR register and asserts the interrupt line, which is shared with DAT1 line in 4-bit mode. The SDHC detects and steers the card's interrupt to the selected IRQ line of the interrupt controller.

34.4.4 SDIO Card Interrupt

34.4.4.1 Interrupts in 1-Bit Mode

In this case, the DAT1 pin is dedicated to providing the interrupt function. Pulling the DAT1 low asserts an interrupt until the host clears the interrupt.

34.4.4.2 Interrupt in 4-Bit Mode

Because the interrupt and data line 1 share pin DAT1 in 4-bit mode, an interrupt is only sent by the card and recognized by the host during a specific time. This is known as the interrupt period. The SDHC samples only the level on DAT1 during the interrupt period. At all other times, the host interrupt controller ignores the level on DAT1. The definition of the interrupt period is different for operations with single block and multiple block data transfers.

In the case of normal single data block transmissions, the interrupt period becomes active two clock cycles after the completion of a data packet. This interrupt period lasts until after the card receives the end bit of the next command with a data block transfer associated with it.

For multiple block data transfers in 4-bit mode there is only a limited period of time the interrupt period can be active due to the limited period of data line availability between multiple blocks of data. This requires a more strict definition of the interrupt period. For this case, the interrupt period is limited to two MMC_SD_CLK clock cycles. This begins two clocks after the end bit of the previous data block. During this 2-clock cycle interrupt period, if an interrupt is pending, the DAT1 line is held low for one clock cycle with the last clock cycle pulling DAT1 high. On completion of the interrupt period, the card releases the DAT1 line into the high impedance state.

When in 4-bit mode, the SDHC differentiates a data start bit and the interrupt period by checking all four data lines are low for the start of new data. In the case of an interrupt, only the DAT1 should have gone low. After the last data block is sent, the interrupt period starts as normal after the end of this data block, but it ends after the next command with a data block commences, instead of lasting two cycles.

Refer to SDIO Card Specification for further information about SDIO card interrupt.

34.4.4.3 Card Interrupt Handling

When the SDIO bit in the interrupt control register is set to 0, the host controller clears the interrupt request to the system interrupt controller. The SDIO Interrupt detection is stopped when this bit is cleared and restarted when this bit is set to 1. The host driver should clear the SDIO interrupt enable bit before servicing the SDIO interrupt and should set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

The SDIO Status bit is cleared by resetting the SDIO interrupt. Writing to this bit has effect in 1-bit mode, as the host controller detects the SDIO interrupt with or without SD clock (to support wakeup). In 4-bit mode, the interrupt signal is sampled during the interrupt period; therefore, some sample delays exist between the interrupt signal from the SDIO card and the interrupt to the host system interrupt controller. When the SDIO status has been set and the host driver needs to start this interrupt service, the SDIO bit in the interrupt control register is set to 0 to clear the SDIO interrupt status latched in the SDHC and to stop driving the interrupt signal to the system interrupt controller. The host driver must issue a CMD52 to clear the interrupts at the card. After completion of the card interrupt service, the SDIO interrupt enable bit is set to 1 and SDHC starts sampling the interrupt signal again. See [Figure 34-21 \(a\)](#) for illustration of the SDIO card interrupt scheme and [Figure 34-21 \(b\)](#) for the sequences of software and hardware events taking place during card interrupt handling procedure

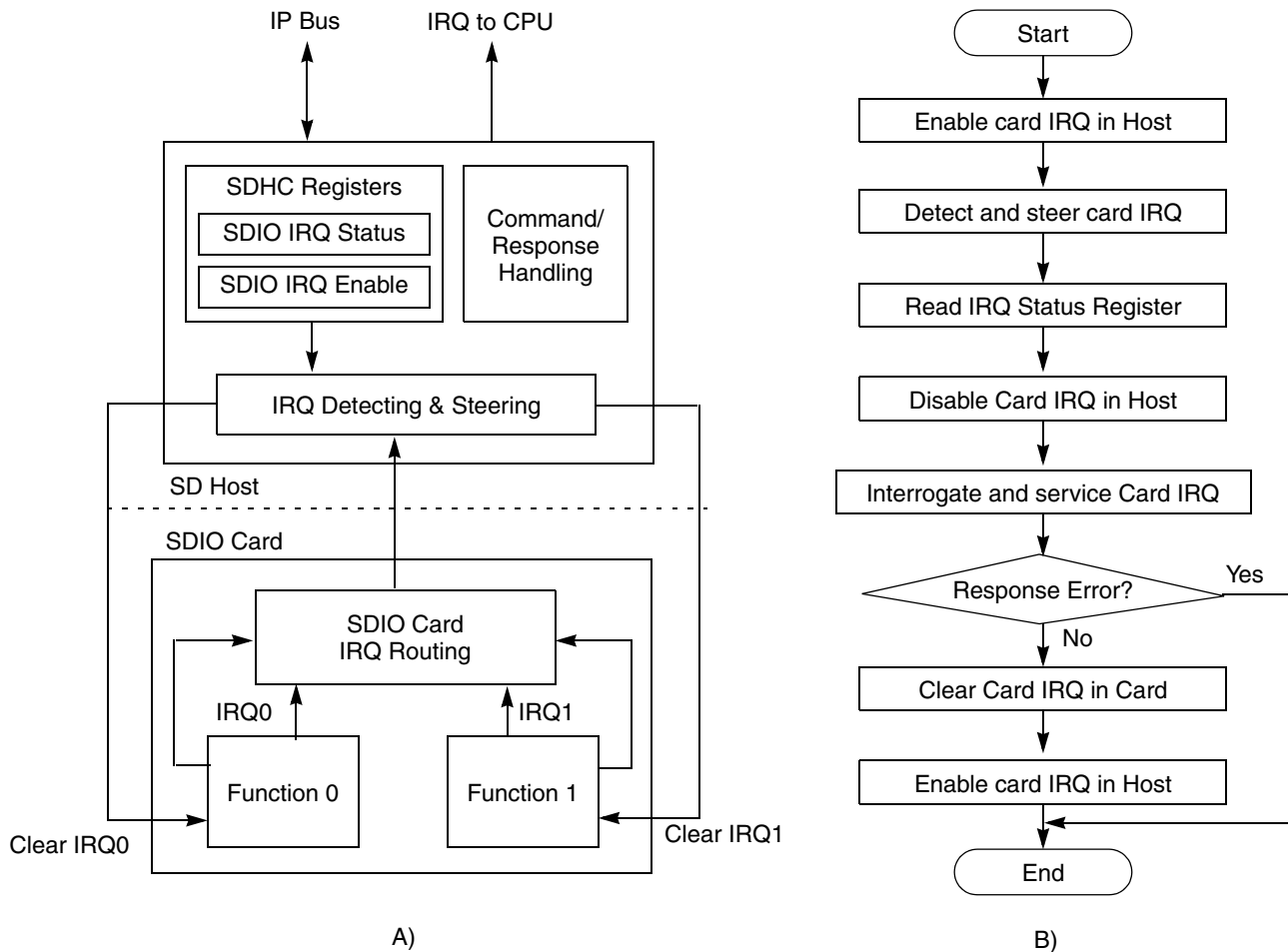


Figure 34-21. A) Card Interrupt Scheme; B) Card Interrupt Detection and Handling Procedure

34.4.5 Card Insertion and Removal Detection

SDHC uses the DAT3 pin to detect card insertion or removal. To use this feature of the SDHC, chip level integration needs to pull-down this pad as a default state. When no card exists on the MMC/SD bus, DAT3 defaults to a low voltage level. When any card is inserted to or removed from the socket, SDHC detects the logic value changes on the DAT3 pin and generates an interrupt.

Because the mechanism is based on the value of the DAT3 line, only single-card systems can benefit from card detection. To avoid conflicts of card insertion/removal detection and the data value changes on DAT3 due to data transfer, disable the card insertion interrupt when a card is detected in the socket and enable it when the card is removed from the socket. The card removal interrupt can only be enabled when no bus activity occurs on DAT3.

To avoid false status bit generation during data transfer, the card insertion/removal is masked by corresponding interrupt enable bit in INT_CNTR register.

Above all, there are three interrupt sources: card removal interrupt, card insertion interrupt, and SDIO card interrupt. All interrupt sources are ORed between the peripheral and the interrupt controller.

NOTE

Send a command (CMD42 for SDMem or CMD52 for SDIO) to the card to disable the card internal pull-up resistor after card detection and identification. Because the SD protocol requires the DAT line must be pulled up for data transfer, disable the host side of the DAT3 pull-down feature and configure it as pull-up. If the card internal pull-up resistor is disabled during this, the card removal interrupt can not be detected through DAT3.

34.4.6 Power Management

When there is no operation between SDHC and the card through SD bus, disable the `ipg_clk` and `SDHC_CLK` in chip level clock control module to save power. When you need to use SDHC to communicate with the card, enable the clock.

34.4.7 System Clock Controller

There is one clock divider and one clock prescaler in SDHC to divide the high frequency input clock `SDHC_CLK` to a lower frequency clock, which most of the SDHC logic can use. See [Figure 34-22](#) for details about clocks used in SDHC. The input clock first goes through a 4-bit divider and then a 12-bit prescaler to generate a clock named `CLK_20M`. This clock is used internally by SDHC and generates the `MMC_SD_CLK`. The `MMC_SD_CLK` to the card has the same clock frequency as `CLK_20M`. `CLK_20M` is derived from the `CLK_DIV` by using the 12-bit prescaler. The `CLK_DIV` is derived from the input clock `SDHC_CLK` by using the 4-bit divider. SDHC clock rate register controls the divide rate for both the divider and the prescaler. Refer to [Section 34.3.2.3, “SDHC Clock Rate Register \(CLK_RATE\),”](#) for the clock rate register information.

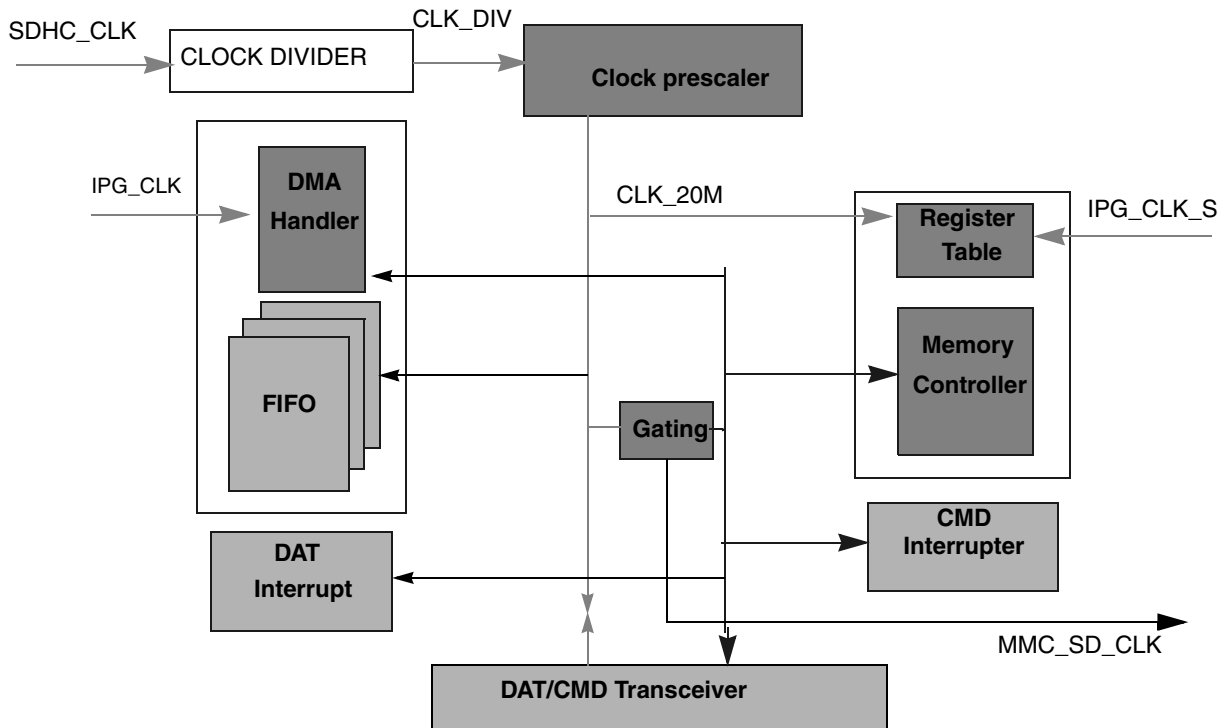


Figure 34-22. Clock Used in SDHC

To get the maximum power-saving during the operation, the SDHC bus clock pauses and resumes according to the SDHC status. For example, when FIFO is full during the card read operation, the bus clock is stopped if no further data is written to FIFO by card. It is resumed when user (DMA) clears FIFO empty status; similarly, there are other conditions where SDHC stops the clock to save power.

The controller controls the rate of the host main clock and checks whether it is on or off. The clock is turned off by setting the bit[0] of the STR_STP_CLK register and is turned on by setting the bit[1] of the STR_STP_CLK. To change the clock rate, the application has to write a new value in the CLK_RATE register.

34.5 Initialization Information

The host controls all communication between system and cards. Also, the host sends commands of two types: broadcast and addressed (point-to-point) commands.

Broadcast commands are intended for all cards, such as: Go_Idle_State, Send_Op_Cond, All_send_CID, and Set_relative_Addr. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. If the socket supports only one card, the broadcast command is similar as the point-to-point command.

After the broadcast command Set_relative_Addr is issued, all cards enter standby mode. Addressed type commands are used from this point on. In this mode, the CMD/DAT I/O returns to push-pull mode to have the driving capability for maximum frequency operation.

As mentioned in the above section, MMC and SD are similar products. Other than the 4x bandwidth, they are programmed similarly. The following example shows how to initialize and perform content access and content protection on the cards.

To improve the readability, use a program-like function for the example.

34.5.1 MMC_SD_CLK Control

STR_STP_CLK register controls the MMC_SD_CLK clock to the card. The clock should be supplied to the card for:

- Submitting command to card and receiving response
- Transferring data between SDHC and the card
- Detecting an interrupt from a SD card in 4-bit

The steps below show how to start the MMC_SD_CLK to card:

1. Write 0x2 to STR_STP_CLK register.
2. Poll STATUS[8], wait until clock starts.

The steps below show how to stop MMC_SD_CLK to card (it is not recommended to stop clock by software):

1. Write 0x1 to STR_STP_CLK register.
2. Poll STATUS[8], wait until clock is stopped.

NOTE

Do not change the ipg_clk_gating_disable and SDHC_CLK_gating_disable bits when start and stop mmcclk.

34.5.2 Command Submit – Response Receive Basic Operation

Below is the program flow to submit a command to the card(s). Targeted command is <command_no>. Corresponding argument is <arg_no>. The command configuration required is <cmd_dat_cont>. The interrupt control used in the user program is <int_Control_value>.

The steps below show how to submit a command to the card:

1. Start MMC_SD_CLK if it is stopped.
2. Enable END_CMD_RESP interrupt by writing 0 to INT_CNTR[2].
3. Set command number to CMD register.
4. Set the command argument to ARG register.
5. Set the appropriate value to Command Data control register (CMD_DAT_CONT).
6. Wait for the command response end interrupt and check for the response CRC/time-out status.
7. Read the response FIFO to check the response. Read three or eight times from the response FIFO access register, depending upon whether the response is 48-bit or 136-bit.

This following is a function defining command submission. This function is used in the examples in the following sub-section:

```
send_cmd_wait_resp(command_no, arg, cmd_dat_cont, int_cntr_value)
{
    write_reg(COMMAND, <command_no>); // 1. configure the CMD
    write_reg(ARG, <arg_no>); // 2. configure the command argument
    write_reg(CMD_DAT_CONT, <cmd_dat_cont>); // 3. configure the command data
    control register, writing to this register triggers SDHC send command to the
    card.
    while(irq_status); // 4. Wait interrupt (End Command Response)
    Write_reg(INT_CNTR, <int_cntr_value>); // 5. irq request from SDHC
    read_reg(STATUS); // 6. Check whether the interrupt is an End_CMD_RES or a
    response time out or a CRC error.
    read_reg(RES_FIFO); // 7. read the response FIFO to determine if the command has
    a response
}
```

34.5.3 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMC cards. All data communication in the card identification mode uses the command line (CMD) only.

34.5.3.1 Card Detect

See [Figure 34-23](#) for a flow diagram showing the card detection using the host controller.

- Write 1 to INT_CNTR[16] to enable card detection interrupt
- Write 0 to INT_CNTR[17] to disable card detection interrupt

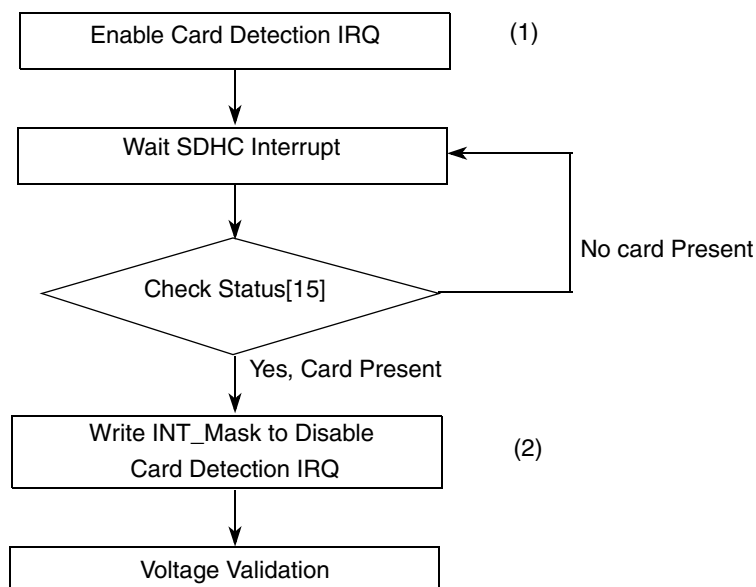


Figure 34-23. Flow Diagram for Card Detection

34.5.3.2 Reset

There are three types of reset:

- Hardware reset (Card and Host) driven by POR (power on reset).
- Software reset (Host Only) proceeds by the write operation on register STR_STP_CLK. Follow the recommended sequence as specified in [Section 34.3.2.3, “SDHC Clock Rate Register \(CLK_RATE\).”](#) The reset can reset all the SDHC registers, but does not reset the card. The card reset is through CMD0. After you apply software reset to SDHC, it should also use CMD0 to reset the card in case the card is in an unknown state. Write 0x2 to register STR_STP_CLK and poll status[8] to wait clock is on. It is highly recommended to start clock here and leave the clock unchanged afterwards. It depends on the specific requirement whether clock automatical gating feature should be enabled (bits 15 and 14 of STR_STP_CLK register).
- Card is reset (Card Only). The command, Go_Idle_State, CMD0 is the software reset command for the MMC and SD memory card. This sets each card into idle state regardless of the current card state. When used as a SD I/O Card, CMD52 writes IO reset in CCCR. The cards are initialized with a default relative card address (RCA=0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See [Figure 34-24](#) for the software flow to reset both SDHC and the card.

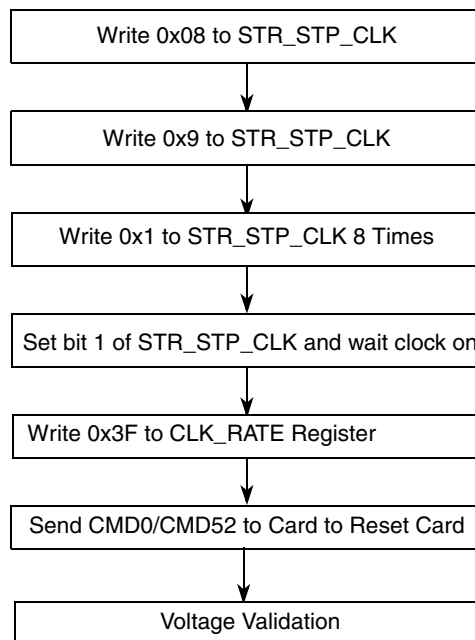


Figure 34-24. Flow Chart for Reset of SDHC and SD I/O Card

```

software_reset()
{
write_reg(STR_STP_CLK, 0x8);
write_reg(STR_STP_CLK, 0x9); // 1. reset the SDHC host;
write_reg(STR_STP_CLK, 0x1);

```

```

write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1);
write_reg(STR_STP_CLK, 0x1); // 2. write 0x1 to STR_STP_CLK 8 times;
write_reg(STR_STP_CLK, 0x2); // 3. write 0x2 to STR_STP_CLK to start clock;
while (!STATUS[8]); // 4. wait clock on
write_reg(CLK_RATE, 0x3F); // 5. Set the lowest clock for initialization
write_reg(READ_TO, 0x2DB4); // 6. set READ timeout register
send_cmd_wait_resp(CMD_GO_IDLE_STATE, 0x0, 0x80, 0x40); // 7. reset the card with CMD0
}

```

34.5.3.3 Voltage Validation

All cards are able to establish communication with the host using any operation voltage in the allowed voltage range specified in the standard. However, the supported minimum and maximum voltages are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID and CSD data in the preload memory can only communicate this information under data transfer Vdd conditions. If the host and card have non-compatible voltage ranges, the card cannot complete the identification cycle nor send CSD data.

Therefore, a special command Send_Op_Cont (CMD1 for MMC), a SD_Send_Op_Cont (CMD41 for SD Memory), and a IO_Send_Op_Cont (CMD5 for SD I/O) provide a mechanism to identify and reject cards which do not match the voltage range desired by the host. The host accomplishes this by sending the required voltage window as the operand of this command. Cards which cannot perform data transfer in the specified range must disable themselves from further bus operations and go into the inactive state. By omitting voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into inactive state. This query should be used if the host can select a common voltage range or if a notification to the application of non-usable cards in the stack is desired.

The following steps show how to perform voltage validation when a card is inserted:

```

voltage_validation(voltage_range_argument)
{
    send_cmd_wait_resp(IO_SEND_OP_COND, 0x0, 0x04, 0x40); // CMD5, send SDIO operation
    voltage, command argument is zero
    if(End Command Response true & No. of IO functions> 0) // it is SDIO and have IO function
    { IORDY = 0;
      while(!(IORDY in I/O ORC response)) { // set voltage range for each IO
        send_cmd_wait_resp(IO_SEND_OP_COND, voltage_range_argument, 0x04, 0x40); }
      if(Memory Present flag true)
        Card = combo; // that is, SDIO + SD Memory, need to set operation voltage to memory
        portion as well
        send_cmd_wait_resp(APP_CMD, 0x0, 0x01, 0x40); // CMD55, Application Command follows
        send_cmd_wait_resp(SD_APP_OP_COND, voltage_range_argument, 0x01, 0x40); // ACMD41
      else
        Card = sdio;
    }
}

```

```

// if No response to CMD5 IO_SEND_OP_COND or No. of IO Function is zero in response
else// the card should be SD or MMC
{send_cmd_wait_resp(APP_CMD, 0x0, 0x01, 0x40); // CMD55, Application Command follows
if(End Command Response true and no response timeout)
{send_cmd_wait_resp(SD_APP_OP_COND, voltage_range_argument, 0x01, 0x40); // ACMD41, SD
card found
Card = sd;
}
else // the card have no response to APP_CMD, it is not SD card
{send_cmd_wait_resp(SEND_OP_COND, voltage_range_argument, 0x01, 0x40); //CMD1, MMC card
found
if(End Command Response true and no response timeout)
{Card = mmc;}
else{ Card = No card or failed contact;}
}
}

```

34.5.3.4 Card Registry

Card registry on MMC and SD cards is different.

For SD Card, the identification process starts at clock rate Fod, initialization clock frequency defined by the card spec, (below 400 kHz for most of the card) as defined by the card spec. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command is sent to all of the new cards in the system. Incompatible cards are put into inactive state. The host then issues the command, All_Send_CID (CMD2), to each card to get its unique card identification (CID) number. Cards currently unidentified (in ready state) send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send_Relative_Addr (CMD3), requesting the card to publish a new relative card address (RCA) shorter than CID. This addresses the card for future data transfer operations. After the RCA is received, the card state changes to the stand-by state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send_Relative_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system.

For MMC operation, the host starts the card identification process in open-drain mode with the identification clock rate Fod. Open drain driver stages on the CMD line allow parallel card operation during card identification. After the bus is activated, the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is a wired operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All_Send_CID (CMD2), asking all cards for their unique card identification (CID) number. All unidentified cards (those in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bitstream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Because CID is unique for each card, only one card can successfully send its full CID to the host. This card then goes into identification state. Thereafter, the host issues Set_Relative_Addr (CMD3) to assign to this card a relative card address (RCA). After the RCA is

received, the card state changes to stand-by state, and the card does not react to further identification cycles. Its output switches from open-drain to push-pull. The host repeats the process, CMD2 and CMD3, until the host receives time-out condition to recognize completion of the identification process.

```
card_registry()
{
while (ResponseTO from STATUS){
if(card==combo or sdio)
{
send_cmd_wait_resp(SET_RELATIVE_ADDR, 0x00, 0x01, 0x40);    //card publish the RCA in
response
rca = SDIO_RCA = address from response FIFO;
}
else if(card==sd)
{
send_cmd_wait_resp(ALL_SEND_CID, 0x00, 0x02, 0x40);
send_cmd_wait_resp(SET_RELATIVE_ADDR, 0x00, 0x01, 0x40); //card publish the RCA in
response
rca = SD_RCA = address from response FIFO;
}
else if(card==mmc)
{
send_cmd_wait_resp(ALL_SEND_CID, 0x00, 0x00, 0x02, 0x40);
rca = MMC_RCA = 0x1;
send_cmd_wait_resp(SET_RELATIVE_ADDR, MMC_RCA_argument, 0x01, 0x40);
}
else
exit due to card not identified;
}
send_cmd_wait_resp(SELECT_CARD, RCA_argument, 0x41, 0x40);
}
```

34.5.4 Card Access

34.5.4.1 Block Access — Block Write & Block Read

34.5.4.1.1 Block Write

During block write, (CMD24 - 27) one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. A card supporting block write can always accept a block of data defined by WRITE_BL_LEN. If the CRC fails, the card indicates the failure on the DAT line (see below); the transferred data is discarded and not written, and all further transmitted blocks (in multiple block write mode) are ignored.

If the host uses partial blocks with accumulated length not block aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card shall detect the block misalignment error and abort programming before the beginning of the first misaligned block. The card sets the ADDRESS_ERROR error bit in the status register, and while ignoring all further data transfer, waits in the receive-data-state for a stop command. The write operation is also aborted if the host tries to write over a write protected area. In this case, however, the card sets the WP_VIOLATION bit.

Programming of the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in ROM, this unchangeable part must match the corresponding part of the receive buffer. If this match fails, the card reports an error and does not change any register contents. Some cards may require unpredictable times to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing and holds the DAT line low if its write buffer is full and unable to accept new data from a new WRITE_BLOCK command. The host may poll the status of the card with a SEND_STATUS command (CMD13) at any time, and the card responds with its status. The status bit READY_FOR_DATA indicates the card can accept new data or the write process remains in progress. The host may deselect the card by issuing CMD7 (to select a different card) which displaces the card into the disconnect state and releases the DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling DAT to low if programming remains in progress and the write buffer is unavailable.

The software flow to write to card with DMA enable is:

1. Start MMC_SD_CLK if it is stopped.
2. Check the card status, wait until card is ready for data.
3. For SD/MMC, set the card block length, using SET_BLOCKLEN (CMD16).
4. Set the SDHC block length register to be same as block length set to the card in Step 3. For SDIO, if the CMD53 is in byte mode, the SDHC block length register should be set according to bytes count in CMD53; if the CMD53 is in block mode, the SDHC block length register should be set according to the block size in CCCR registers.
5. Set SDHC number block register (NOB), nob is 1 for single block write or CMD53 in byte mode for SDIO
6. Disable the buffer ready interrupt, configure the DMA setting and enable the SDHC DMA channel:
 - Write 1 to bit[3] of INT_MASK register in SDHC.
 - Set DMA destination to be SDHC_Buffer Access register.
 - Set DMA destination port size to be 32-bit.
 - Set DMA Burst length to be 16 bytes in 1-bit mode or 64 bytes in 4-bit mode.
 - Set DMA transfer count to be number of bytes which is a multiple of the Block_length($nob * blk_len = \text{total number of bytes}$).
7. Check the card status and wait until the card is ready for data.
8. Set SDHC CMD register to any of the following:
 - CMD24(WRITE_BLOCK), or
 - CMD25(WRITE_MULTIPLE_BLOCK), or
 - CMD53 in byte mode or block mode
9. Set SDHC CMD Argument register.
10. Set SDHC Command Data Control register.
11. Wait for end command response and check if there any CRC error or timeout error.
12. Wait for DMA done.
13. Check for Write_OP_DONE and check status bit to see if write CRC error occurred.

14. Send STOP_TRANSMISSION command to card if the write command is WRITE_MULTIPLE_BLOCK (CMD25).

If the write operation is without DMA, the system needs to write data to the buffer through buffer write ready interrupt or by polling the buffer write ready status bit (STATUS[6]: BUF_WR_RDY). For high performance, data transfer using DMA is preferred.

34.5.4.1.2 Block Read

For block reads, the basic unit of data transfer is a block whose maximum size is defined in the CSD (READ_BL_LEN). If READ_BL_PARTIAL is set, smaller blocks whose starting and ending address are entirely contained within one physical block (as defined by READ_BL_LEN) may also be transmitted. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17 (READ_SINGLE_BLOCK) initiates a block read. After completing the transfer, the card returns to the transfer state. CMD18 (READ_MULTIPLE_BLOCK) starts a transfer of several consecutive blocks. Blocks are continuously transferred until a stop command is issued. If the host uses partial blocks with accumulated length not block aligned and block misalignment is not allowed, the card detects a block misalignment at the beginning of the first mis-aligned block, sets the ADDRESS_ERROR error bit in the status register, aborts transmission, and waits in the data state for a STOP command.

The software flow to write to card with DMA enable is:

1. Start MMC_SD_CLK if it is stopped.
2. Check the card status and wait until the card is ready for data.
3. For SD/MMC, set the card block length, using SET_BLOCKLEN (CMD16).
4. Set the SDHC block length register to be same as block length set to the card in Step 3.
For SDIO, if the CMD53 is in byte mode, the SDHC block length register should be set according to bytes count in CMD53; if the CMD53 is in block mode, the SDHC block length register should be set according to the block size in CCCR registers.
5. Set SDHC number block register (NOB) to 1 for single block write or CMD53 in byte mode for SDIO.
6. Disable the buffer ready interrupt, configure the DMA setting, and enable the SDHC dma channel:
 - Write 0 to bit[4] of INT_CNTR register in SDHC to disable the buffer read ready interrupt.
 - Set DMA source to be SDHC_Buffer Access register.
 - Set DMA source port size to be 32-bit.
 - Set DMA Burst length to be 16 bytes in 1-bit mode or 64 bytes in 4-bit mode.
 - Set DMA transfer count to be number of bytes which is a number of blocks multiple of the Block_length(nob*blk_len).
7. Check the card status and wait until the card is ready for data.
8. Set SDHC CMD register to be CMD17(READ_SINGLE_BLOCK) or CMD18 (READ_MULTIPLE_BLOCK) or CMD53 in byte mode or block mode.
9. Set SDHC CMD Argument register.
10. Set SDHC Command Data Control register

11. Wait for END_CMD_RESP interrupt and check response FIFO, check CRC error and timeout error.
12. Wait for DMA done.
13. Check for READ_OP_DONE and check status bit to see if read CRC error occurred.
14. Send STOP_TRANSMISSION command to card if the read command is READ_MULTIPLE_BLOCK (CMD18).

If the read transfer operation does not use DMA, the system needs to fetch data out of the data buffer through utilizing the buffer read ready interrupt or by polling the buffer read ready status bit (STATUS[7]: BUF_READ_RDY). For high performance, data transfer using DMA is preferred.

34.5.5 Switch Card Mode

Switch function command (CMD6) switches or expands memory card functions. Two function groups are defined:

- Card access mode - 12.5MB/sec interface speed (default) or 25MB/sec interface speed. (highspeed).
- Card command system - Standard command set (default) or eCommerce command set or Vendor Specific Command set.

This is a new feature, introduced in SD Specifications Part 1 PHYSICAL LAYER Specification Version 1.10, so cards compatible with earlier version does not support this feature. Before issuing CMD6 to switch-card mode, the host driver checks SD_SPEC field in SCR register to confirm the command is supported. Card only accepts CMD6 in transfer state. After selected, all functions only return to the default function after a power cycle, CMD6 (Mode 1 operation with Function 0 in each function group) or CMD0. Executing a power cycle or issuing CMD0 causes the card to reset to the idle state and all the functions to switch back to the default function. On responding to CMD6, the card sends R1 response on the CMD line and 512 bits of status on the DAT lines. Therefore, for the host controller, this is like CMD17 for a single block read with block size of 64 bytes. The time-out value of this command is also 100ms. If CRC error occurs on the status data, the host driver should issue a power cycle. CMD6 function switching period is within eight clocks after the end bit of status data. When CMD6 changes the bus behavior, the host can use the new functions then.

The software flow to enable high speed mode with DMA enabled is (assume SD_SPEC field is verified):

1. Start MMC_SD_CLK if it is stopped.
2. Check the card status and wait until the card is ready for data.
3. Set the card block length to 64 bytes, using SET_BLOCKLEN (CMD16).
4. Set SDHC number block register (NOB) to 1.
5. Disable the buffer ready interrupt, configure the DMA setting, and enable the SDHC DMA channel:
 - Write 0 to bit[4] of INT_CNTR register in SDHC to disable the buffer read ready interrupt.
 - Set DMA source to be SDHC_Buffer Access register.
 - Set DMA source port size to be 32-bit.

- Set DMA burst length to be 16 bytes in 1-bit mode or 64 bytes in 4-bit mode.
- Set DMA transfer count to be 64 bytes.
- 6. Check the card status and wait until the card is ready for data.
- 7. Set SDHC CMD register to be CMD6(SWITCH).
- 8. Set SDHC CMD Argument register to 0xFFFFF1.
- 9. Set SDHC Command Data Control register.
- 10. Wait for END_CMD_RESP interrupt and check response FIFO, check CRC error and timeout error.
- 11. Wait for DMA done.
- 12. Check for READ_OP_DONE and check status bit to see if read CRC error occurred.
- 13. Check if bit 401 of received 512 bits is 1 to confirm high speed mode is supported.
- 14. Repeat steps 6~12 except in step 8, set CMD Argument as 0x80FFFFFF1
- 15. Check if bits 379~376 are 4'b0001 of received 512 bits to confirm high speed mode is enabled.

Change clock to about 50 MHz for high speed mode.

Chapter 35

Software Watchdog Timer (WDT)

35.1 Introduction

The Software Watchdog Timer (WDT) provides a method to recover from conditions caused by improper software operation. For example, software may become lost due to a programming error or an electrical problem. Also, the software may become trapped in a loop with no controlled exit.

The WDT is a down-counter that is periodically reset to a maximum count by writing a special service sequence to the software watchdog service register (SWSRR). If the WDT decrements to 0x0000, a software reset or a machine check processor exception (MCP) is generated. The WDT is enabled or disabled at the release of reset, depending upon the state of the software watchdog enable bit (SWEN) of the reset configuration word high register, whose value is determined by the state of an external configuration pin. The SWEN bit of the reset configuration word high register is generally set as 1, so the WDT is default enabled at the release of reset. The WDT can be disabled by setting the SWEN bit to 0. The SWEN bit is in the watchdog control register (SWCRR).

35.1.1 Features

Key features of the WDT include the following:

- 16-bit prescaler and 16-bit down-counter
- Selectable range for timeout period
- Timeout delay of approximately 128 seconds maximum with a 33 MHz system XTAL clock¹

35.1.2 Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:
If the WDT is not needed, the user can disable it with software after a system reset. When it is disabled, the watchdog counter and prescaler counter are held in a stopped state.
- WDT output reset/interrupt mode:
Without periodic software servicing, the WDT times out and issues a reset or a nonmaskable MCP interrupt
- WDT prescaled/non-prescaled clock mode:
The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit that controls the divide-by-65,536 of the WDT prescaler counter.

¹ The system XTAL clock can be different from 33 MHz.

35.2 Memory Map/Register Definition

The WDT programmable register map occupies 16 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

It is recommended that user software not access undefined or reserved locations in the programmable register map for the WDT or any other module. Some locations in the programmable register map may be designated as undefined or reserved, but in fact, may contain non-user mode registers used for testing or for modifying the operation of the module.

All WDT registers are 16- or 32-bits wide, located on 16-bit address boundaries, and should be accessed as 16-bit or 32-bit quantities. That is, 16-bit wide registers should be addressed using half-word accesses, and 32-bit wide registers should be addressed using word accesses. All addresses used in this chapter are offsets from the WDT base address, as defined in [Section 2.2, “Memory Map and Register Definition”](#).

35.2.1 Memory Map

A memory map of the WDT is shown in [Table 35-1](#).

Table 35-1. WDT Register Address Map

Offset	Register	Access	Section/ Page
0x0 – 0x3	Reserved	—	—
0x4	Software watchdog control register (SWCRR)	R/W	35.2.2.1/35-2
0x8	Software watchdog count register (SWCNR)	R	35.2.2.2/35-4
0xC – 0xD	Reserved	—	—
0xE	Software watchdog service register (SWSRR)	R/W	35.2.2.3/35-5

35.2.2 Register Descriptions

35.2.2.1 Software Watchdog Control Register (SWCRR)

The software watchdog control register (SWCRR), shown in [Figure 35-1](#), controls the software watchdog period and configures WDT operation. The SWCRR can be read at any time but can be only written once after system reset.

Offset 0x4Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SWTC															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	SWEN	SWRI	SWPR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	S ¹	1	1

1. Special—SWCRR[SWEN] reset value directly depends on RCWH[SWEN] (reset configuration word high).

Figure 35-1. Software Watchdog Control Register (SWCRR)

Table 35-2. SWCRR Field Descriptions

Field	Description
SWTC	Software watchdog time count. The SWTC field contains the modulus that is reloaded into the watchdog counter by the special service sequence. When a new value is loaded into SWCRR[SWTC], WDT is not updated until the special service sequence is written to the SWSRR register. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count. The new value is also used at the next and all subsequent reloads. Reading the SWCRR register returns the value in the SWCRR. Reset initializes the SWCRR[SWTC] field to 0xFFFF. Note: The prescaler counter is reset each time the contents of the SWTC field is loaded into the watchdog counter due to executing a special service sequence or by executing a system reset.
SWEN	Software watchdog enable. The SWCRR[SWEN] bit enables the WDT. This bit can be cleared by software after a system reset to disable the WDT. When the timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. 0 WDT disabled 1 WDT enabled
SWRI	Software watchdog reset/interrupt select. This bit determines whether a WDT time out causes a hardware reset or machine check interrupt to the core. 0 WDT causes an MCP interrupt to the core 1 WDT causes a hardware reset
SWPR	Software watchdog counter prescale bit. Controls the divide-by-65,536 WDT counter prescaler. 0 The WDT counter is not prescaled 1 The WDT counter clock is prescaled

35.2.2.2 Software Watchdog Count Register (SWCNR)

The software watchdog count register (SWCNR), shown in Figure 35-2, provides visibility to the watchdog counter value. SWCNR is a read-only register. Writing to the SWCNR register has no effect and terminates without transfer error exception.

Offset 0x08Access: User read only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SWCN															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	= Unimplemented or Reserved															

Figure 35-2. Software Watchdog Count Register (SWCNR)

Table 35-3. SWCNR Field Descriptions

Field	Description
SWCN	Software watchdog count. The read-only SWCNR[SWCN] field reflects the current value in the watchdog counter. Writing to the SWCNR register has no effect, and write cycles are terminated normally. Reset initializes the SWCNR[SWCN] field to 0xFFFF. Note: Reading the 16 LS bits of 32-bit SWCNR register with two 8-bit reads is not guaranteed to return a coherent value. Always use a half-word or word access to read this register.

35.2.2.3 Software Watchdog Service Register (SWSRR)

The software watchdog service register (SWSRR) is shown in Figure 35-3. After the WDT is enabled, it must be periodically reset. This is accomplished in applications software by writing the special service sequence of 0x556C followed by 0xAA39 to the SWSRR register before the watchdog counter decrements to 0x0000. If the SWSRR register is not serviced before the timeout, the watchdog timer generates a system reset or MCP interrupt.

Both writes must occur before the timeout in the order listed; however, any number of instructions can be executed between the two writes. Writing any value other than 0x556C or 0xAA39 to the SWSRR register resets the servicing sequence. Both values must be written to keep the watchdog timer from decrementing to 0x0000. Reset initializes the SWSRR[WS] field to 0x0000. SWSRR can be written at any time, but returns all zeros when read.

Offset 0xEAccess: User write only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	WS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-3. Software Watchdog Service Register (SWSRR)

Table 35-4. SWSRR Field Descriptions

Field	Description
WS	Software watchdog service. The user must write 0x556C followed by 0xAA39 to this register to prevent WDT timeout. SWSRR[WS] can be written at any time, but returns all zeros when read.

35.3 Functional Description

35.3.1 Software Watchdog Timer Unit

In the case where the software becomes trapped in loops with no controlled exit, the MPC5121e platform provides the WDT option to prevent system lock. Watchdog timer operations are configured in the software watchdog control register (SWCRR).

The WDT is defaultly enabled after reset (if reset configuration RCWHR[SWEN] is active) to cause a hardware reset or non-maskable MCP interrupt if the WDT decrements to 0x0000. If the WDT is not needed, the user must clear the SWCRR[SWEN] bit to disable it. If used, the WDT requires a special service sequence to be executed periodically. Without this periodic servicing, it times out and issues a reset or a nonmaskable MCP interrupt, as programmed in SWCRR[SWRI]. After software writes to the SWRI bit, the state of SWEN cannot be changed.

The WDT service sequence consists of the following two steps:

1. Write 0x556C to the SWSRR
2. Write 0xAA39 to SWSRR

This special service sequence reloads the WDT, and the timing process begins again. If a value other than 0x556C or 0xAA39 is written to the SWSRR, the entire sequence must start over. Although the writes must occur in the correct order before a timeout, any number of instructions can be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. Figure 35-4 shows a state diagram for the WDT.

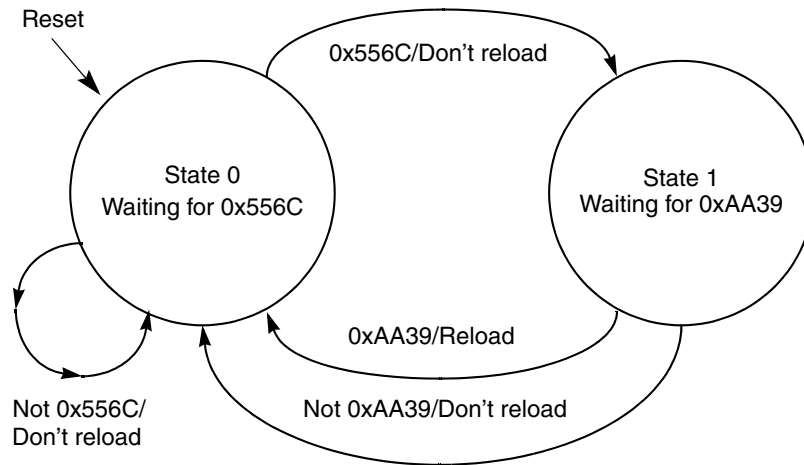


Figure 35-4. Software Watchdog Timer Service State Diagram

Although most software disciplines permit or even encourage the watchdog concept, some systems require a selection of timeout periods. For this reason, the WDT must provide a selectable range for the timeout period. Figure 35-5 shows how to manage this need.

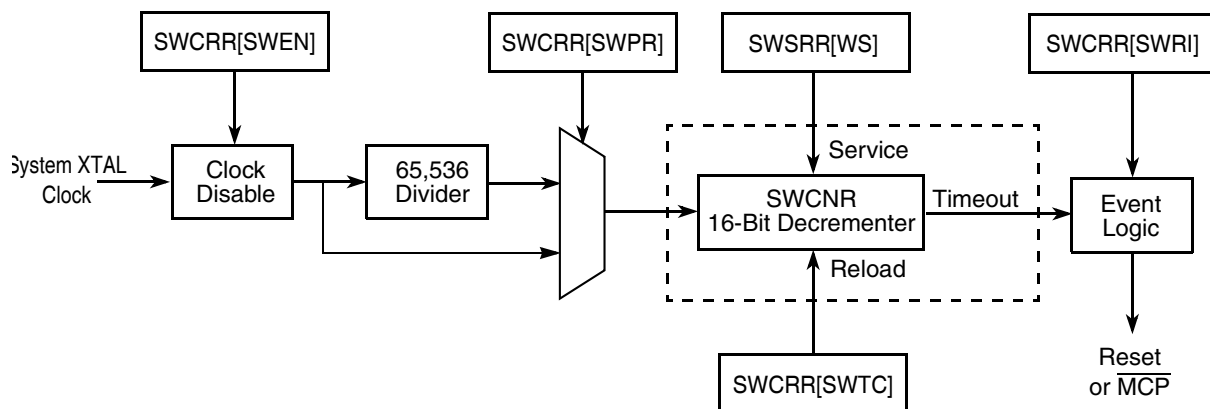


Figure 35-5. Software Watchdog Timer Functional Block Diagram

In [Figure 35-5](#), the range is determined by the value of the SWCRR[SWTC] field. The value in SWTC is loaded into a 16-bit decremter clocked by the system XTAL clock. An additional divide-by-65,536 prescaler value is used when needed.

The decremter begins counting when loaded with a value from SWTC. After the timer reaches 0x000, a software watchdog expiration request is issued to the reset or MCP control logic. Upon reset, the SWTC field is set to the maximum value and is again loaded into the software watchdog count register (SWCRR), starting the process over. When a new value is loaded into SWTC, the WDT is not updated until the servicing sequence is written to the SWSRR. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count.

35.3.2 Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode

If the WDT is not needed, the user can disable it. The SWCRR[SWEN] bit enables the watchdog timer. It should be cleared by software after a system reset to disable the WDT. When it is disabled, the watchdog counter and prescaler counter are held in a stopped state.

 - WDT enable mode (SWCRR[SWEN] = 1)

This is the default value after soft reset.
 - WDT disable mode (SWCRR[SWEN] = 0)

If the WDT is not needed, the user must clear SWCRR[SWEN] to disable it.
- WDT reset/interrupt output mode

Without periodic software servicing, the WDT times out and issues a reset or a nonmaskable MCP interrupt.

According to SWCRR[SWRI] programming, WDT causes a hardware reset or MCP interrupt to the core.

 - Reset mode (SWCRR[SWRI] = 1)

WDT causes a hardware reset.
 - Interrupt mode (SWCRR[SWRI] = 0)

WDT causes an MCP interrupt to the core.
- WDT prescaled/non-prescaled clock mode

The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit that controls the divide-by-65,536 of the WDT counter.

 - Prescale mode (CRR[SWPR] = 1)

The WDT clock is prescaled.
 - Non-prescale mode (CRR[SWPR] = 0)

The WDT clock is not prescaled.

35.3.2.1 WDT Enable/Disable Mode

From the release of RESET, the WDT is defaultly enabled (if reset configuration RCWHR[SWEN] is active). If the WDT is not needed, the user can disable it by clearing the SWCRR[SWEN] bit. The SWCRR[SWEN] bit is cleared by the software after a system reset to disable the WDT. When it is disabled, the watchdog counter and prescaler counter are held in a stopped state. The SWCRR register is a write-once register; therefore, only software can disable the WDT. After it is disabled, it can only be re-enabled by resetting the system.

If the WDT is not needed, the user must clear the SWCRR[SWEN] bit to disable it.

35.3.2.2 WDT Reset/Interrupt Output Mode

Without periodic software servicing, the WDT times out and issues a reset or a nonmaskable MCP interrupt.

If the SWRI bit of the SWCRR is clear when the WDT times out, an MCP interrupt to the CPU core is created. If the SWRI bit is set when the WDT times out, a hardware reset is created.

35.3.2.3 WDT Prescaled/Non-Prescaled Clock Mode

The WDT counter clock can be prescaled by 1 or 65,536, depending up the setting of the prescale bit of the SWCRR.

Prescale mode (SWCRR[SWPR] = 1) — The WDT counter clock uses a prescaler of 65,536.

Non-prescale mode (SWCRR[SWPR] = 0) — The WDT counter clock uses a prescaler of 1.

Chapter 36

Sony/Philips Digital Interface (SPDIF)

36.1 Introduction

The Sony/Philips Digital Interface (SPDIF) audio module is a stereo transceiver that allows the MPC5121E to receive and transmit digital audio over it. The MPC5121E provides a single SPDIF receiver with one input, and one SPDIF transmitter with one output. The SPDIF transceiver allows the handling of both SPDIF channel status (CS) and User (U) data and bows a frequency measurement block that exists to allow precise measurement of an incoming sampling frequency.

Figure 36-1 displays a block diagram of the SPDIF transceiver (both receiver and transmitter) data paths and interface.

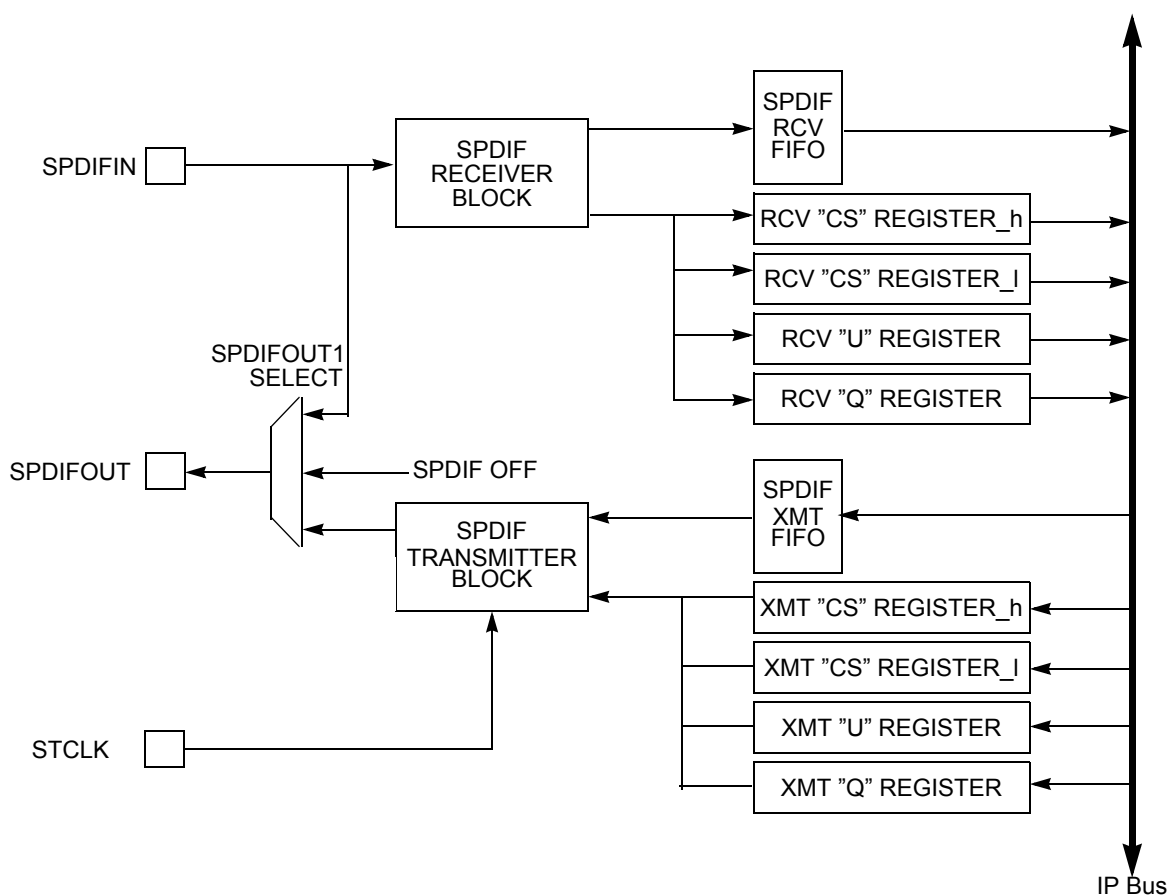


Figure 36-1. SPDIF Transceiver Data Interface Block Diagram

36.1.1 Features

- SPDIF Receiver
 - Single channel receiver
 - Input sample rate measurement
 - 32kHz, 44.1kHz, 48kHz, 64kHz, 88.2kHz, and 96kHz support
 - CS and U bit Recovery
- SPDIF Transmitter
 - Single SPDIF Transmitter
 - One SPDIF output
 - CD Subcode Support
 - C and U bit Support
- SPDIF Receiver to SPDIF Transmitter Bypass Mode

36.2 External Signal Description

36.2.1 Pin Signal Descriptions

Table 36-1 displays the properties of SPDIF external pins signals.

Table 36-1. Signal Properties

Name	Function	I/O	Reset
SPDIFOUT	SPDIF Output, sends audio and non-audio data	O	0
SPDIFIN	SPDIF Input, receives audio and non-audio data	I	—
STCLK	SPDIF transmit clock input	I	—

36.2.2 Detailed Signal Descriptions

contains the detailed descriptions of SPDIF external pins signals.

Table 36-2. Detailed Signal Descriptions

Signal	Description
SPDIFOUT	SPDIFOUT sends audio and non-audio data in the IEC958 formats in a biphase mark format. The output signal is unbalanced, but may support consumer digital transmission formats. Data sent from the SPDIFOUT pin may be generated from either the SPDIF transmitter or redirected from any of the SPDIF input pins (SPDIFIN) as selected by the Txsel field of the EBU_ConfigReg register. The field description of EBU_ConfigReg is shown in Table 36-4.
SPDIFIN	SPDIFIN is used for receiving audio and non-audio data in the IEC958 formats in a biphase mark format.
STCLK	SPDIF transmit clock input pin. After frequency divided by the factor TxClk_DF field in Tx_Div_Reg register, it is put into use as the EBU_OutClock to transmit SPDIF output. The field description of Tx_Div_Reg is shown in Table 36-22.

36.3 Memory Map and Register Definition

Table 36-3 displays the SPDIF memory-mapped 32-bit registers, described in [Section 36.3.1, “Register Descriptions”](#). The bit 0 is MSB, bit 31 is LSB.

Table 36-3. SPDIF Memory Map

Address SPDIF_BAS+	Access	Name	Description	Section/Page
0x00	R/W	EBU_ConfigReg (SCR)	IEC958 configuration register	36.3.1.1/36-4
0x04	R/W	CDTEXT_Control (SRCD)	CDText configuration register	36.3.1.2/36-6
0x08	R/W	PhaseConfig (SRPC)	FreqMeas configuration register	36.3.1.3/36-7
0x0C	R/W	InterruptEn (SIE)	Interrupt enable register	36.3.1.4.1/36-8
0x18	R-Stat W-Clear	InterruptStat/Clear (SIS/SIC)	Interrupt status/clear register	36.3.1.4.2/36-10
0x10	R	EbuRcvLeft (SRXL)	EBU receive data - left channel	36.3.1.5.1/36-13
0x14	R	EbuRcvRight (SRXR)	EBU receive data - right channel	36.3.1.5.2/36-14
0x1C	R	EBU_RxCChannel_h (SRCSH)	EBU receive C channel, bits [47:24]	36.3.1.5.3/36-15
0x20	R	EBU_RxCChannel_l (SRCSL)	EBU receive C channel, bits [23:0]	36.3.1.5.4/36-16
0x24	R	EBU_RxUChannel (SRU)	EBU receive U channel	36.3.1.5.5/36-17
0x28	R	EBU_RxQChannel (SRQ)	EBU receive Q channel	36.3.1.5.6/36-18
0x2C	R	FreqMeas(SRFM)	FreqMeasurement	36.3.1.6/36-18
0x30	W	EbuTxLeft (STXL)	EBU transmit Left channel	36.3.1.7.1/36-20
0x34	W	EbuTxRight (STXR)	EBU transmit Right channel	36.3.1.7.2/36-21
0x38	R/W	EBU_TxCChannelCons_h (STCSH)	EBU transmit Cons. C channel, bits [47:24]	36.3.1.7.3/36-22
0x3C	R/W	EBU_TxCChannelCons_l (STCSL)	EBU transmit Cons. C channel, bits [23:0]	36.3.1.7.4/36-23
0x48	R/W	EBU_TxUChannel (STU)	EBU transmit U channel	36.3.1.7.5/36-24
0x50	R/W	Tx_Div_Reg (STC)	Transmit clock control register	36.3.1.8/36-25

36.3.1 Register Descriptions

36.3.1.1 SPDIF Configuration Register

The SPDIF transceiver has a 32-bit control register EBU_ConfigReg as follows.

Offset 0x00 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RcvFIFO_Ctrl	RcvFIFO_Off/On	RcvFIFO_Rst	RcvFIFOFull_Set	RcvAutoSync	TxAutoSync	TxFIFOEmpty_Set						TxFIFO_Ctrl	PDIR_Rcv	PDIR_Tx	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			ValCtr						0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-2. EBU_Config Register (SCR)
(Note that the register is repeated for reference.)

Table 36-4. SCR Field Descriptions

Field	Description
RcvFIFO_Ctrl	0: Normal operation 1: Always read zero from rcv data register
RcvFIFO_Off/On	0: SPDIF Rcv FIFO is on 1: SPDIF Rcv FIFO is off. Does not accept data from interface
RcvFIFO_Rst	0: Normal operation 1: Reset register to 1 sample remaining
RcvFIFOFull_Set	00: Full interrupt if at least 1 sample in FIFO 01: Full interrupt if at least 4 samples in FIFO 10: Full interrupt if at least 8 samples in FIFO 11: Full interrupt if at least 12 samples in FIFO
RcvAutoSync	0: Rcv FIFO auto sync off 1: Rcv FIFO auto sync on
TxAutoSync	0: Tx FIFO auto sync off 1: Tx FIFO auto sync on
TxFIFOEmpty_Set	00 or 11: Empty interrupt if tx FIFO empty 01: Empty interrupt if less than or equal 4 sample in FIFO 10: Empty interrupt if less than or equal 8 sample in FIFO
TxFIFO_Ctrl	00: Send out digital zero on SPDIF Tx 01: Normal operation 10: Reset to 1 sample remaining 11: Reserved

Offset 0x00Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RcvFIFO_Ctrl	RcvFIFO_Off/On	RcvFIFO_Rst	RcvFIFOFull_Set	Rcv Auto Sync	Tx Auto Sync	TxFIFOEmpty_Set						TxFIFO_Ctrl	PDIR_Rcv	PDIR_Tx	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R			ValCtrl	TxSel			USrc_Sel		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-2. EBU_Config Register (SCR)
(Note that the register is repeated for reference.)

Table 36-4. SCR Field Descriptions (continued)

Field	Description
PDIR_Rcv	0: Not to send DMA Request when PDIR1 FIFO full (Full condition is set by RcvFIFOFull_Set) 1: Send DMA Request when PDIR1 FIFO full
PDIR_Tx	0: Not to send DMA Transmit Request when Transmit FIFO empty (Empty condition is set by TxFIFOEmpty_Set) 1: Send DMA Transmit Request when Transmit FIFO empty
ValCtrl	0: Outgoing Validity always set 1: Outgoing Validity always clear
TxSel	000: Off and output 0 001: Feed-through SPDIFIN 101: Normal operation Others: Reserved
USrc_Sel (U channel source select)	00: No embedded U channel 01: U channel from SPDIF receive block (CD mode) 10: Reserved 11: U channel from on chip transmitter

36.3.1.2 CDText Control Register

The CDText_Control register is associated with User channel reception control as follows.

Offset 0x04 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	PrSetCount						
W									PrSet En							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0		U	U	0	0	0	0	0	0	0	0
W							Sync Mode	Chan TxTim								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-3. CDText_Control Register (SRCD)

Table 36-5. SRCD Field Descriptions

Field	Description
PrSetEn ¹	0: No action on free-running sync position counter 1: Preset free-running sync position counter
PrSetCount	Presetting frame counter value for sync position. It could be set the value between 0-97.
USyncMode	0: Non-CD data 1: CD user channel subcode
UChanTxTim	0: Reserved, since no cd-text output interface 1: Timing to reg. UChannelTx from SPDIF output interface

¹ On read back, zero is returned.

36.3.1.3 PhaseConfig Register (SRPC)

The PhaseConfig Register includes the information of coef selection and clock source selection for frequency measurement as follows.

Offset 0x08 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	LOCK	CoefSel				ClkSrc_Sel				0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-4. PhaseConfig Register (SRPC)

Table 36-6. SRPC Field Descriptions

Field	Description
LOCK	LOCK bit to show the internal DPLL is locked, read only
CoefSel	The coeffect value is used to adjust the result of frequency measure. Make the result value more fitter to a 32-bit register. CoefSel Coeffect value: 3'b000: 24 3'b001: 16 3'b010: 12 3'b011: 8 3'b100: 6 3'b101: 4 Others: 3
ClkSrc_Sel	Clock source selection: 000: if (DPLL Locked) SPDIF_RcvClk else STClk 011: STClk Others: Reserved

36.3.1.4 Interrupt Registers

The interrupt registers include InterruptEn, InterruptStat and InterruptClear. The InterruptEn register provides control over the enabling of interrupts. The InterruptStat register is a read only register providing status on interrupt operation. The InterruptClear register is a write only register and is used to clear interrupts.

36.3.1.4.1 InterruptEn Register (SIE)

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	TxUnOv	TxResyn	CNew	ValNoGood	SymErr	BitErr	UTxEEm	UTxUnder	UTx_NextFirst	URxFul	URxOv	QRxFul
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	QRxOv	UQSync	UQErr	PdirUnOv	PdirResyn	LockLoss	TxEEm	PdirFul	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-5. InterruptEn Register (SIE)
(Note that the register is repeated for reference.)

Table 36-7. SIE Field Descriptions

Field	Description
TxUnOv	SPDIF transmit FIFO under/overflow
TxResyn	SPDIF transmit FIFO resync
CNew	SPDIF receive change in value of control channel
ValNoGood	SPDIF validity flag no good
SymErr	SPDIF receiver found illegal symbol
BitErr	SPDIF receiver found parity bit error
UTxEEm	UChannel transmit register empty, this bit can't be cleared with reg. IntClear. To clear it, write to U Tx reg.
UTxUnder	UChannel transmit register underrun
UTx_NextFirst	UChannel transmit register next byte is first, this bit can't be cleared with reg. IntClear. To clear it, write to U Tx reg.
URxFul	UChannel receive register full, this bit can't be cleared with reg. IntClear. To clear it, read from U Rcv reg.
URxOv	UChannel receive register overrun
QRxFul	QChannel receive register full, this bit can't be cleared with reg. IntClear. To clear it, read from Q Rcv reg.
QRxOV	QChannel receive register overrun
UQSync	U/Q Channel sync found

Offset 0x0CAccess: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	TxUnOv	TxResyn	CNew	ValNoGood	SymErr	BitErr	UTxEm	UTxUnder	UTxNextFirst	URxFul	URxOv	QRxFul
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	QRxOv	UQSync	UQErr	PdirUnOv	PdirResyn	LockLoss	TxEEm	PdirFul	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-5. InterruptEn Register (SIE)
 (Note that the register is repeated for reference.)

Table 36-7. SIE Field Descriptions (continued)

Field	Description
UQErr	U/Q Channel framing error
PdirUnOv	Processor data input underrun/overrun
PdirResyn	Processor data input resync
LockLoss	SPDIF receiver loss of lock
TxEEm	SPDIF transmit FIFO empty, this bit can't be cleared with reg. IntClear. To clear it, write to Tx FIFO.
PdirFul	Processor data input full, this bit can't be cleared with reg. IntClear. To clear it, read from PDIR FIFO.

36.3.1.4.2 InterruptStat Register (SIS)

Offset 0x18 Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	TxUnOv	TxResyn	CNew	ValNoGood	SymErr	BitErr	UTxEm	UTxUnder	UTx_NextFirst	URxFul	URxOv	QRxFul
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	QRxOv	UQSync	UQErr	PdirUnOv	PdirResyn	LockLoss	TxEEm	PdirFul	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-6. InterruptStat Register (SIS)

Table 36-8. SIS Field Descriptions

Field	Description
TxUnOv	SPDIF transmit FIFO under/overflow
TxResyn	SPDIF transmit FIFO resync
CNew	SPDIF receive change in value of control channel
ValNoGood	SPDIF validity flag no good
SymErr	SPDIF receiver found illegal symbol
BitErr	SPDIF receiver found parity bit error
UTxEm	UChannel transmit register empty, this bit can't be cleared with reg. IntClear. To clear it, write to U Tx reg.
UTxUnder	UChannel transmit register underrun
UTx_NextFirst	UChannel transmit register next byte is first, this bit can't be cleared with reg. IntClear. To clear it, write to U Tx reg.
URxFul	UChannel receive register full, this bit can't be cleared with reg. IntClear. To clear it, read from U Rcv reg.
URxOv	UChannel receive register overrun
QRxFul	QChannel receive register full, this bit can't be cleared with reg. IntClear. To clear it, read from Q Rcv reg.
QRxOV	QChannel receive register overrun
UQSync	U/Q Channel sync found
UQErr	U/Q Channel framing error
PdirUnOv	Processor data input underrun/overflow
PdirResyn	Processor data input resync
LockLoss	SPDIF receiver loss of lock
TxEEm	SPDIF transmit FIFO empty, this bit can't be cleared with reg. IntClear. To clear it, write to Tx FIFO.

Offset 0x18 Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	TxUn Ov	Tx Resyn	CNew	ValNo Good	Sym Err	BitErr	UTx Em	UTx Under	UTx_ Next First	URx Ful	URx Ov	QRx Ful
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	QRx Ov	UQ Sync	UQ Err	Pdir UnOv	Pdir Resyn	Lock Loss	TxEEm	Pdir Ful	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-6. InterruptStat Register (SIS)

Table 36-8. SIS Field Descriptions (continued)

Field	Description
PdirFul	Processor data input full, this bit can't be cleared with reg. IntClear. To clear it, read from PDIR FIFO.

36.3.1.4.3 InterruptClear Register (SIC)

Offset 0x18 Access: User write only

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0												
W					TxUnOv	TxResyn	CNew	ValNoGood	SymErr	BitErr		UTxUnder			URxOv	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R									0	0	0	0	0	0	0	0
W	QRxOv	UQSync	UQErr	PdirUnOv	PdirResyn	LockLoss										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-7. InterruptClear Register (SIC)

Table 36-9. SIC Field Descriptions

Field	Description
TxUnOv	SPDIF transmit FIFO under/overflow
TxResyn	SPDIF transmit FIFO resync
CNew	SPDIF receive change in value of control channel
ValNoGood	SPDIF validity flag no good
SymErr	SPDIF receiver found illegal symbol
BitErr	SPDIF receiver found parity bit error
UTxUnder	UChannel transmit register underrun
URxOv	UChannel receive register overrun
QRxOV	QChannel receive register overrun
UQSync	U/Q Channel sync found
UQErr	U/Q Channel framing error
PdirUnOv	Processor data input underrun/overflow
PdirResyn	Processor data input resync
LockLoss	SPDIF receiver loss of lock

36.3.1.5 SPDIF Reception Registers

SPDIF reception registers include audio data reception registers: EbuRcvLeft and EbuRcvRight, channel status reception registers: EBU_RxCCChannel_h and EBU_RxCCChannel_l, user bits reception registers: EBU_RxUChannel and EBU_RxQChannel.

36.3.1.5.1 EbuRcvLeft Register (SRXL)

Offset 0x10Access: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RcvDataLeft															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RcvDataLeft								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-8. EBU_RcvLeft Register (SRXL)

Table 36-10. SRXL Field Descriptions

Field	Description
RcvDataLeft	Processor receive SPDIF data left

36.3.1.5.2 EbuRcvRight Register (SRXR)

Offset 0x14Access: User read

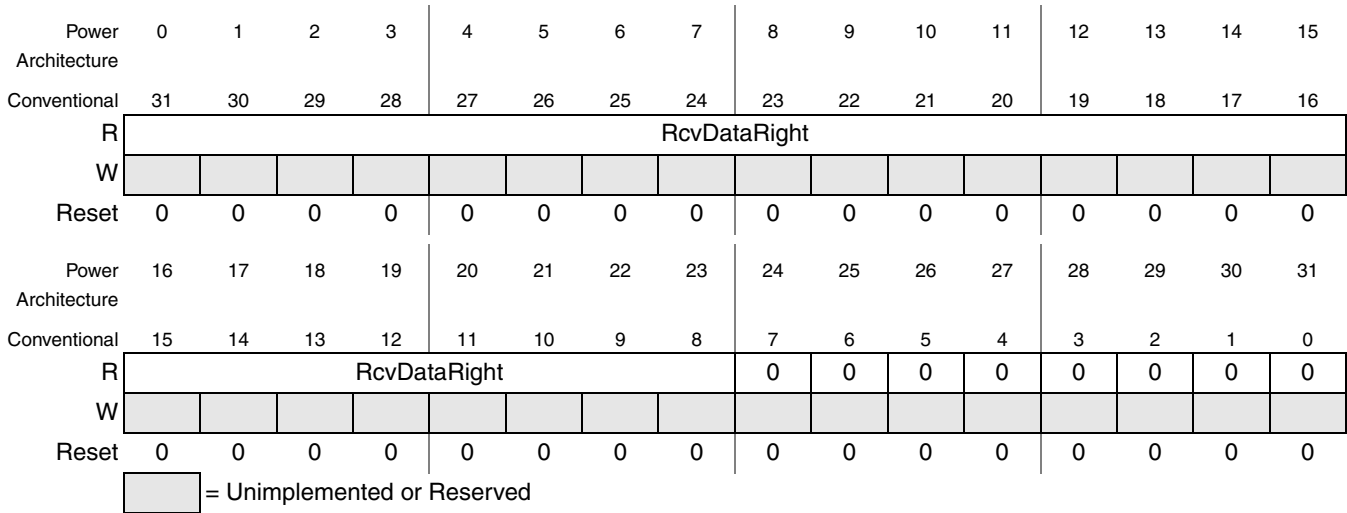


Figure 36-9. EbuRcvRight Register (SRXR)

Table 36-11. SRXR Field Descriptions

Field	Description
RcvDataLeft	Processor receive SPDIF data right

36.3.1.5.3 EBU_RxCChannel_h Register (SRCSH)

Offset 0x1CAccess: User read

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RxCChannel_h															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RxCChannel_h								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-10. EBU_RxCChannel_h Register (SRCSH)

Table 36-12. SRCSH Field Descriptions

Field	Description
RxCChannel_h	SPDIF receive C channel register, contains first 24 bits of C channel without interpretation

36.3.1.5.4 EBU_RxCChannel_I Register (SRCSL)

Offset 0x20Access: User read

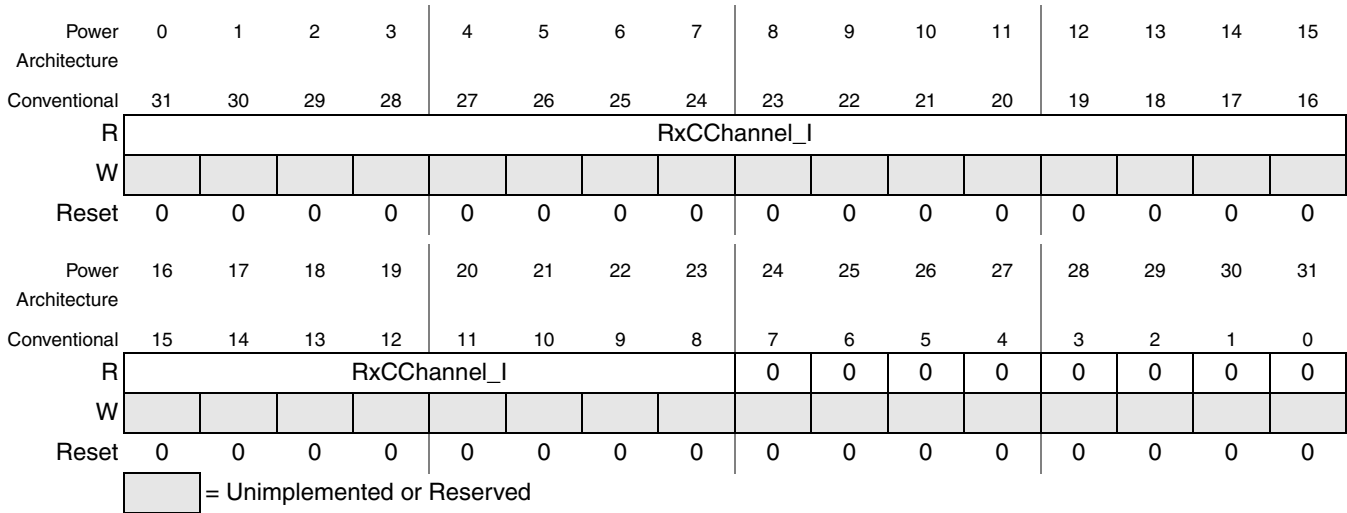


Figure 36-11. EBU_RxCChannel_I Register (SRCSL)

Table 36-13. SRCSL Field Descriptions

Field	Description
RxCChannel_I	SPDIF receive C channel register, contains next 24 bits of C channel without interpretation

36.3.1.5.5 EBU_RxUChannel Register (SRU)

Offset 0x24 Access: User read

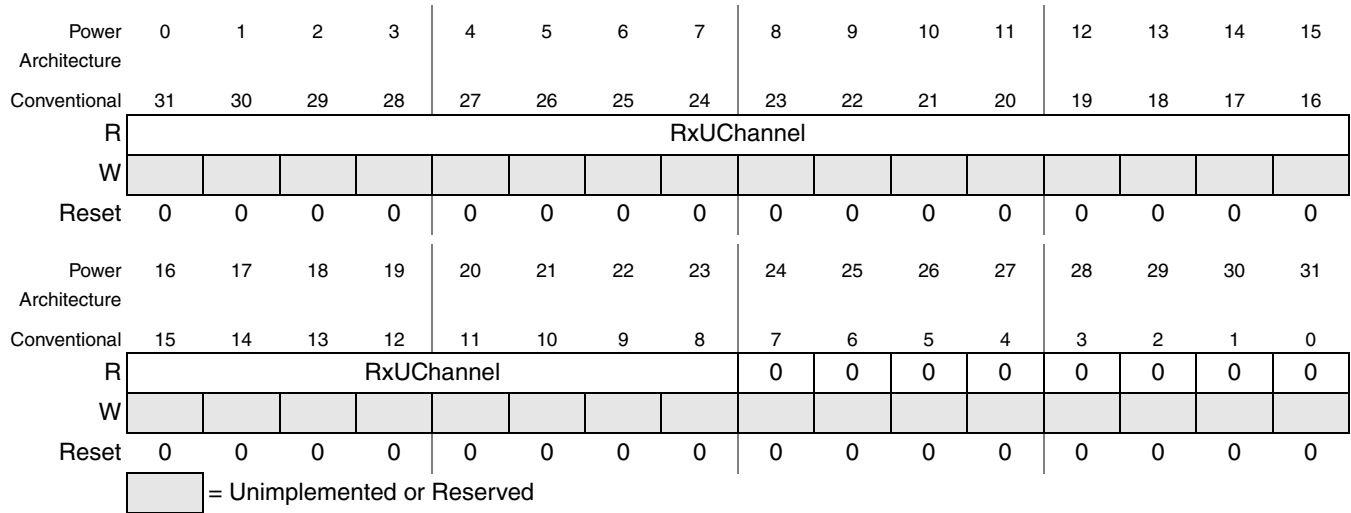


Figure 36-12. EBU_RxUChannel Register (SRU)

Table 36-14. SRU Field Descriptions

Field	Description
RxUChannel	SPDIF receive U channel register, contains next 3 U channel bytes

36.3.1.5.6 EBU_RxQChannel (SRQ)

Offset 0x28Access: User read

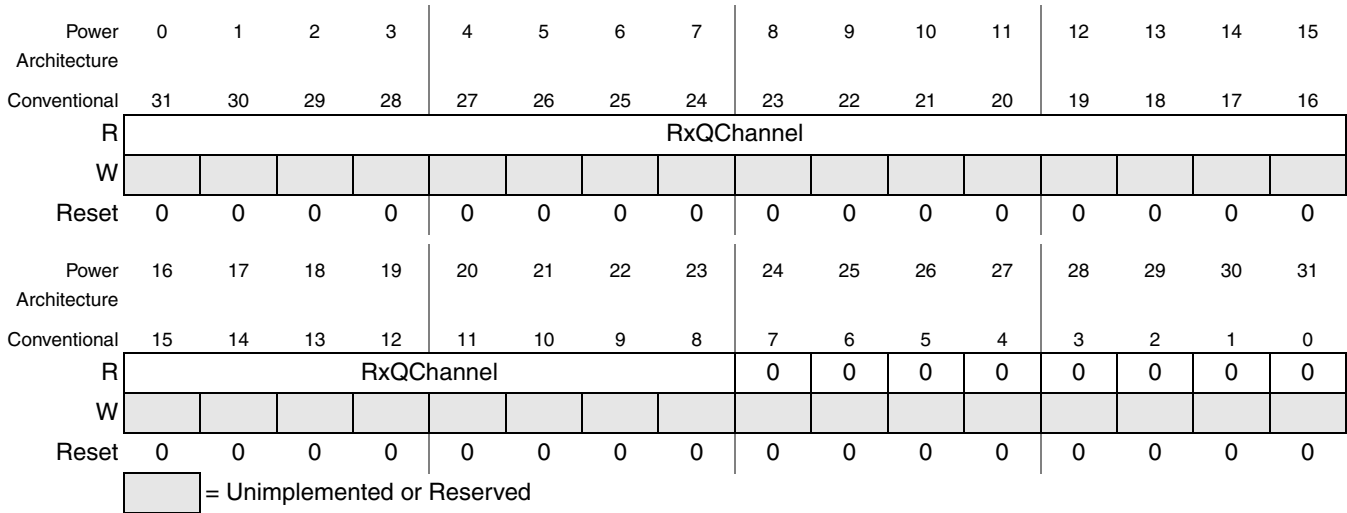


Figure 36-13. EBU_RxQChannel Register (SRQ)

Table 36-15. SRQ Field Descriptions

Field	Description
RxQChannel	SPDIF receive Q channel register, contains next 3 Q channel bytes

36.3.1.6 FreqMeas Register (SRFM)

This register is used to save the result of frequency measurement of the input source. It is read only.

Offset 0x2CAccess: User read

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FreqMeas															
W																
Reset	0	0	0	0	0	0	0	0	0	0						
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FreqMeas								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-14. FreqMeas Register (SRFM)**Table 36-16. SRFM Field Descriptions**

Field	Description
FreqMeas	A ratio of measured clock frequency to reference clock(IPG_CLK) frequency: $(\text{FreqMeas_CLK}/\text{IPG_CLK}) * (2^{**20}) * \text{COEF}$

36.3.1.7 SPDIF Transmission Registers

SPDIF transmission registers include audio data transmission registers: EbuTxLeft and EbuTxRight, channel status transmission registers: EBU_TxCCChannelCons_h, EBU_TxCCChannelCons_l and EBU_TxCCChannelProf, user bits transmission register: EBU_TxUChannel.

36.3.1.7.1 EbuTxLeft Register (STXL)

Offset 0x30Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	TxDataLeft															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	TxDataLeft								0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-15. EbuTxLeft Register (STXL)

Table 36-17. STXL Field Descriptions

Field	Description
TxDataLeft	SPDIF transmit left channel data. It is write-only, and always returns zeros when read

36.3.1.7.2 EbuTxRight Register (STXR)

Offset 0x34 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	TxDataRight															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	TxDataRight															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-16. EbuTxRight Register (STXR)

Table 36-18. STXR Field Descriptions

Field	Description
TxDataRight	SPDIF transmit right channel data. It is write-only, and always returns zeros when read.

36.3.1.7.3 EBU_TxCChannelCons_h Register (STCSH)

Offset 0x38Access: User read/write

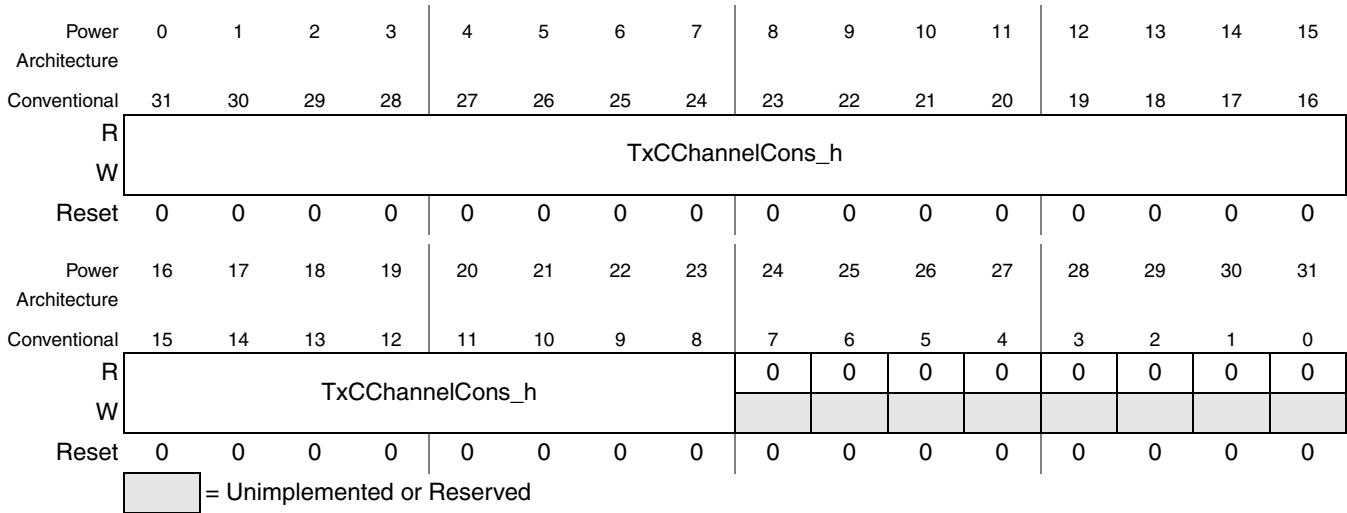


Figure 36-17. EBU_TxCChannelCons_h Register (STCSH)

Table 36-19. STCSH Field Descriptions

Field	Description
TxCChannel Cons_h	SPDIF transmit Cons. C channel data, contains next 24 bits without interpretation. When read, it returns the latest data written by the processor.

36.3.1.7.4 EBU_TxCChannelCons_I Register (STCSL)

Offset 0x3cAccess: User read/write

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TxChannelCons_I															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TxChannelCons_I								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 36-18. EBU_TxCChannelCons_I Register (STCSL)

Table 36-20. STCSL Field Descriptions

Field	Description
TxCChannelCons_I	SPDIF transmit Cons. C channel data, contains next 24 bits without interpretation. When read, it returns the latest data written by the processor.

36.3.1.7.5 EBU_TxUChannel Register (STU)

Offset 0x48Access: User read/write

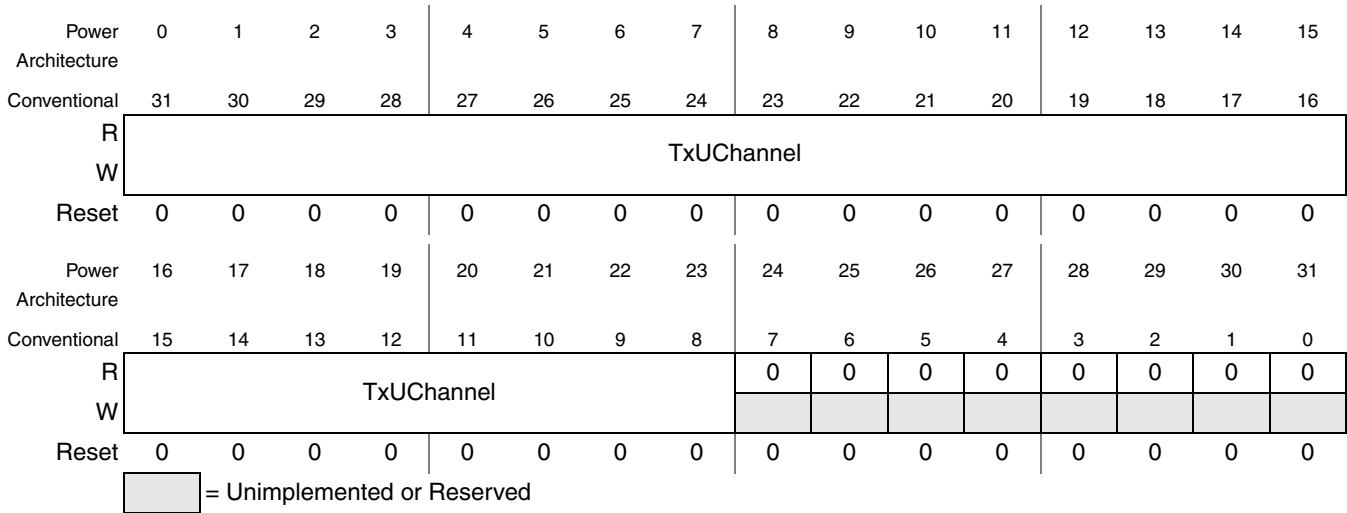


Figure 36-19. EBU_TxUChannel Register (STU)

Table 36-21. STU Field Descriptions

Field	Description
TxUChannel	SPDIF transmit U channel register, contains next 3 U channel bytes. When read, it returns the latest data written by the processor.

36.3.1.8 Transmit Clock Control Register (STC)

This register includes the information of transmit clock selection and frequency division as follows.

Offset 0x50 Access: User read/write

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TxClk_Sel	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	TxClk_DF				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 36-20. Transmit Clock Control Register (STC)

Table 36-22. STC Field Descriptions

Field	Description
TxClk_Sel	00: Select STClk others: Reserved
TxClk_DF	Divider factor (1-16)

The TxClk_Df field setting divides the TxClk by any integer between 1 and 16. The duty cycle is always 1:1. The relation between the value of TxClk_Df field and divide factor is list as follow:

4'b0000 - divide by 1 (not be divided)

4'b0001 - divide by 2

4'b0010 - divide by 3

...

4'b1111- divide by 16

36.4 Functional Description

36.4.1 SPDIF Receiver

The SPDIF receiver extracts the audio data from each SPDIF frame and places the data in a 12-deep FIFO. The Channel Status and User Bits are also extracted from each frame and placed in corresponding registers. The SPDIF receiver also provides a bypass option for direct transfer of the SPDIF input signal to the SPDIF transmitter.

The SPDIF receiver handles the main data audio stream and recovers the bit clock from the SPDIF input signal. The sample rate can be determined from the frequency measuring block. Additionally, the receiver supports the SPDIF C and U channels. SPDIF C and U channel data is interfaced directly to memory-mapped registers. The input data is sent via a 12-deep FIFO to the memory-mapped data registers. All the data are controlled by the Interrupt Control Block and transferred to memory-mapped IP bus.

The following functions are performed by the SPDIF receiver.

- Audio Data Reception
- Channel Status bits Reception
- User Channel bits Reception
- Validity Flag Reception
- SPDIF Receiver Exception support
- SPDIF Lock Detection

36.4.1.1 Audio Data Reception

The SPDIF Receiver block extracts the audio data from the IEC958 stream, and outputs this via a 12-deep FIFO to the memory-mapped registers EbuRcvLeft and EbuRcvRight. Data from the SPDIF receiver is buffered in receive FIFO, and can be read by the processor from the memory-mapped registers.

The audio data is received as 24 bits, and the memory-mapped registers are 32 bits. When the audio data is sent to the memory-mapped registers, MSB is aligned and the low 8 bits of EbuRcvLeft and EbuRcvRight are always read as zeros.

36.4.1.1.1 SPDIF Receiver Data Registers – Behavior on Overrun/Underrun

The SPDIF Data Receive registers (EbuRcvLeft and EbuRcvRight) have different FIFOs for left and right channel. As a result, there is always the possibility that left and right FIFOs may go out of sync due to FIFO underruns and FIFO overruns that affect only one part (left or right) of any FIFO. To prevent this from happening, hardware has been added on the device. Two mechanisms to prevent mismatch between the FIFOs are available.

If a SPDIF Data Receive FIFO overrun occurs on e.g. the right half of the FIFO, the sample that caused the overrun is not written to the right half (due to overrun). Special hardware makes sure the next sample is not written to the left half of the FIFO. If the overrun occurs on the left half of the FIFO, next sample is not written to the right half of the FIFO.

36.4.1.1.2 SPDIF Receiver Data Registers—Automatic Resynchronization of FIFOs

An automatic FIFO resynchronization feature is available. It can be enabled and disabled separately for every FIFO. If enabled, the hardware checks if the left and right FIFO are in sync. If not, it sets the filling pointer of the right FIFO to be equal to the filling pointer of the left FIFO.

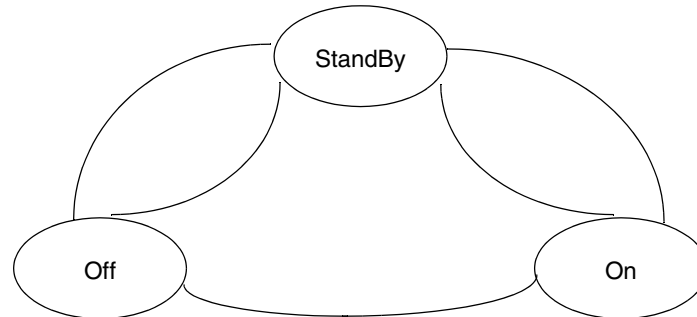


Figure 36-21. FIFO Auto-Resync Controller State Machine

The operation can be explained from the state diagram in [Figure 36-21](#). Every FIFO auto-resync controller has a state machine with three states: Off, StandBy and On. In the On state, the filling of the left FIFO is compared with the filling of right, and if they are not equal, right is made equal to left, and an interrupt is generated.

The controller stays in the off state when the feature is disabled. When not disabled, the state machine moves to off state on any processor read or write to the FIFO. It moves from On or Off to Standby on any left sample read from SPDIF Tx FIFOs, or on any left sample write to SPDIF Rcv FIFOs. The controller moves from Standby to On on any right sample read from SPDIF Tx FIFO, or on any right sample write to SPDIF Rcv FIFO. There is a control bit in the SPDIFConfig register to enable/disable the feature for the SPDIF Rcv FIFO and SPDIF Tx FIFO.

36.4.1.2 Application Note

The automatic FIFO resynchronization can be switched on and avoids all mismatches between left and right FIFOs, if the software obeys the following rules: 1. When the left data is read or written to the left FIFO, in the same place of the program, data must be read or written to the right FIFO. Maximum time difference between left and right is 1/2 sample clock. (E.g. if sample frequency is 44 KHz, approx 10 micro-seconds. For 88 KHz, approx 5 micro-seconds.) 2. Write/read data to FIFOs at least two samples at the time. If there is a mismatch Left-Right, the resync logic may go on only 1 sample clock after last data is read/written to the FIFO. Also acceptable is polling the FIFO, if at least part of the time two samples are read/written to it.

36.4.1.2.1 SPDIF Receiver—Additional Features

There are three exceptions associated with the SPDIF Receivers FIFOs.

- Full
- Under/Overflow
- Resync

When the full condition (which is set by RcvFIFOFull_Set field of the EBU_ConfigReg) is set for processor data input registers, the processor should read data from the FIFO, before overrun occurs. For example, when full condition is set while the FIFO contains 12 samples, it is acceptable for the software to read first 12 samples from the LEFT address, followed by 12 samples from the RIGHT address, or 12 samples from the RIGHT address, followed by 12 samples from the LEFT address, or 1 sample LEFT, followed by 1 sample RIGHT repeated 12 times. There is no order specified.

The implementation for SPDIFRcv is a double FIFO, one for left and one for right. Full is set when both FIFOs are full. Underrun and overrun are set when one of the FIFOs do underrun or do overrun. The resync interrupt means hardware took special action to resynchronize left and right FIFOs.

The FIFO level at which the full interrupt is generated, is programmable via the Full Select field in the EBU_ConfigReg register.

36.4.1.2.2 Rcv FIFO On and Rcv FIFO Reset

Two additional control fields of the SPDIF Rcv FIFO are the on/off select and FIFO reset fields.

If on/off select is set to off, all-zero is read from the FIFO, irrespective of the data received over the SPDIF interface.

If FIFO reset is set, the FIFO is blocked at 1 sample in FIFO. In this, the full interrupt is on if FullSelect is set to 00. If FullSelect is set to any other value, interrupt is off. The other interrupts are always off.

36.4.1.3 Channel Status Reception

A total of 48 channel status bits are received in two registers. No interpretation is performed by the SPDIF receiver module. Channel Status Bits are ordered first bit left. CS-channel MSB bit 0 is located in bit position 0 in the memory-mapped register EBU_RxCCChannel_h. CS-channel bit 23 is considered the LSB bit 23 in the register. C-channel bit 24 to 47 is seen as [0:23] bits of register EBU_RxCCChannel_l. Because the two memory-mapped registers are 32 bits, 8 zeros are filled in bits field [24:31].

36.4.1.4 Channel Status interrupt

When the value of a new SPDIF CS channel status frame is loaded in the register, an interrupt is generated. The interrupt is cleared when the processor writes the corresponding bit in the InterruptStat register.

36.4.1.5 User Bit Reception

There are two modes for User Channel reception, CD and non-CD. As is decided by USyncMode (bit 22 of CDText_Control register).

36.4.1.5.1 Behavior of User Channel Receive Interface on Incoming CD User Channel Sub-code in SPDIF Receiver

This mode is selected if UsyncMode, bit 22 in register CD Text control is set to 1.

The CD subcode stream embedded into the SPDIF User channel consists of a sequence of packets. Every packet is made up 98 symbols. The first two symbols of every packet are sync symbols, the other 96 symbols are data symbols.

Any sequence found in the SPDIF U-channel stream starting with a leading one, followed by 7 information bits, is recognized as a data symbol. Subsequent data symbols are separated by pauses. During the pause, zero bits are seen on the SPDIF U-channel.

Data symbols are coming in MSB first. The MSB is the leading one.

When a long pause is seen between two subsequent data symbols, the SPDIF receiver assumes the reception of one or more sync symbols.

Table 36-23. Sync Control Bits

Number of U Channel Zero Bits	Corresponding Number of Sync Symbols
0-1	Unpredictable, not allowed
2-10	0
11-22	1
23-34	2
35-46	3
>45	Unpredictable, not allowed

The recognition of the number of sync symbols derives from the fact that the U-channel transmitter in the CD channel decoder transmits one symbol on average every 12 SPDIF channel bits. On this average rate, there is a tolerance of maximum 5 %.

The SPDIF receiver is tolerant on symbol error. Due to the physical nature of the transmission of the data over the CD disc, not more than one out of any 5 consecutive user channel symbols may be in error. The error may cause a change in data value, which is not treated by this interface, or it may cause a data symbol to be seen as a sync symbol, or a sync symbol to be seen as a data symbol. However, not more than one out of any 5 consecutive user channel symbols should be affected in this way.

The SPDIF User channel circuitry recognizes the 98-symbol packet structure, and sent the 96 symbol payload to the processor application. The 96 symbol payload is transmitted to the processor via 2 registers:

- The EBU_RxUChannel register. In this register, data is presented 3 symbols at the time to the processor. Every time 3 new valid symbols, received on the SPDIF U-Channel are present, the UChannelRcvFull interrupt is asserted. For one 98-symbol packet, 96 symbols are carried across EBU_RxUChannel. To transfer all this data, 32 UChannelRcvFull interrupts are generated.
- The QChannelReceive register. In this register, only the Q bit of the packet is accumulated. Operation is similar to UChannelReceive. Because only Q-bit is transferred, only 96

Q-bits are transferred for any 98-symbol packet. To transfer this data, 4 QChannelRcvFull interrupts are generated. When QChannelRcvFull occurs, it is coincident with UChannelRcvFull. There is only one QChannelRcvFull for every 8 UChannelRcvFull. The convention is that most significant data is transmitted first, and is left-aligned in the registers.

- Timing regarding packet boundary is extracted by hardware. The last UChannelRcv-Full corresponding to a given packet should be coincident with the last QChannel-RcvFull. In this last U, Q channel interrupt, symbols 95-98 are received, Q-channel bits 67-98. The interrupts are coincident with UQSyncFound, flagging last symbols of the current frame.
- When the start of the new packet is found before the current packet is complete (less than 98 symbols in the packet), the UQFrameError interrupt is set. The application software should read out UChannelReceive and QchannelReceive registers, discard the value, and assume the start of a new packet.
- As already said, packet sync extraction is tolerant for single-symbol errors. Packet sync detection is based on the recognition of the sequence data-sync-sync-data in the symbol stream, because this is the only syncing sequence that is not affected by single errors. If the sync symbols are not found 98 symbols after the previous occurrence, it is assumed to be destroyed by channel error, and a new sync symbols is interpolated.
- Normally, only data bytes are passed to the application software. Every databyte has its most significant bit set. If sync symbols are passed to the application soft, they are seen as all-zero symbols. Sync symbols can only end up in the data stream due to channel error.

36.4.1.5.2 Behavior of User Channel Receive Interface on Incoming Non-CD Data

This mode is selected if UsyncMode, bit 22 in register CD Text control is set 0.

In non-CD mode, the SPDIF User channel stream is recognized as a sequence of data symbols. No packet recognition is done.

Any sequence found in the SPDIF U-channel stream starting with a leading one, followed by 7 information bits, is recognized as a data symbol. Subsequent data symbols are separated by pauses. During the pause, zero bits are seen on the SPDIF U-channel.

Three consecutive data symbols seen in the SPDIF U-Channel stream are grouped together into the EBU_RxUChannel register. First symbol is left, last symbol is right aligned. When EBU_RxUChannel contains 3 new data symbols, UChannelRcvFull is asserted.

In this mode, the operation of QchannelRcv and associated interrupt QchannelRcvFull is reserved, undefined. Also reserved, undefined is operation of UQFrameError and UQSyncFound.

The U-channel is extracted, and output by the SPDIF rcv block on SPDIFRcvUChannel-Stream.

When incoming SPDIF data parity error or bit error is detected, and if the next SPDIF word for that channel is error-free, the SPDIF word in error is replaced with the average of the previous word and next word. When incoming SPDIF data parity error or bit error is detected, and the next SPDIF word is in error, the previous SPDIF word is repeated.

36.4.1.6 Validity Flag Reception

An interrupt is associated with the Validity flag. (interrupt 7 – SPDIFValNoGood). This interrupt is set every time a frame is seen on the SPDIF interface with the validity bit set to invalid.

36.4.1.7 SPDIF Receiver Interrupt Exception Definition

There are several SPDIF exceptions defined, that triggers an interrupt.

These are:

- Control Status channel change. Set when SPDIFRcvCChannel1 register is updated. The register is updated for every new C-Channel received. The exception is reset on write to InterruptClear register.
- SPDIF Illegal Symbol. Set on reception of illegal symbol during SPDIF receive. Reset by writing register InterruptClear.SPDIF bit error. Set on reception of bit error. Parity bit does not match. Reset on write to InterruptClear register.
- Receive data FIFO full. Set when SPDIF receive data FIFO is full.
- Receive data FIFO underrun/overflow. Set when there is a underrun/overflow on the SPDIF receive data FIFO.
- Receive data FIFO resynchronization. Set when a resynchronization event occurs on the SPDIF receive data FIFO.
- Receive U Channel buffer full. Set when next 24 bits of U channel code are available.
- Receive Q Channel buffer overflow. Set when Q channel buffer overflow.
- Receive U Channel buffer overflow. Set on U channel buffer overflow.
- Receive Q Channel buffer full. Set when next 24 bits of Q channel code are available.
- Receive UQ sync found. Set when UQ channel sync found.
- Receive UQ frame error. Set when UQ frame error found.

36.4.1.8 Standards Compliancy

The SPDIF interface is compatible with the Tech 3250-E standard of the European Broadcasting Union, except clause 6.3.3 and the IEC958-3 Ed2 for relevant topics. Supported input frequency range is 12 KHz up to 96 KHz. (fully compliant) and 96 KHz up to 176 KHz (can interface with compliant SPDIF transmitter within same cabinet, making reasonable assumptions on jitter added due to interconnecting wire).

The SPDIF input is a biphase/mark modulated signal. The time between any two successive transitions of the SPDIF signal is always 1, 2 or 3 SPDIF symbol periods long. The SPDIF receiver parses the stream, and split it in so-called symbols. It recognizes s1, s2 and s3 symbols, depending on the length of the symbols. Not all sequences of these symbols are allowed. To give an example, a sequence s2-s1-s1-s1-s2 cannot occur in a no-error SPDIF signal. If the receiver finds such an illegal sequence, the illegal symbol interrupt is set. No corrective action is undertaken. When the interrupt occurs, this means that the SPDIF signal is destroyed by noise and the SPDIF frequency changed.

Tolerated jitter on SPDIF input signals are 0.25 bit peak-peak for high frequencies. There is no jitter limit for low frequencies. The user channel extraction in CD mode is capable of coping with single-symbol

errors and retrieve U-channel frames on correct boundaries. This capability is required for reliable reception of CD-Text from some Philips CD channel decoders. This capability was deemed more important than compliance with the IEC958 annex A.3 standard, and for this reason user channel reception is not compliant with IEC958 annex A.3. However, the interface is capable to receive U channel inserted by a typical CD channel decoder. Also, in this case, it is more robust and tolerant for channel error than what is required by IEC958 annex A.3.

36.4.1.9 Measuring Frequency of SPDIF_RcvClk

The internal DPLL can extract the bit clock (advanced plus) from the input bitstream. It is necessary, however, to measure the frequency of the incoming signal in relationship with the clock BUS_CLK (In the SPDIF is IPG_CLK). The circuit is shown in [Figure 36-22](#).

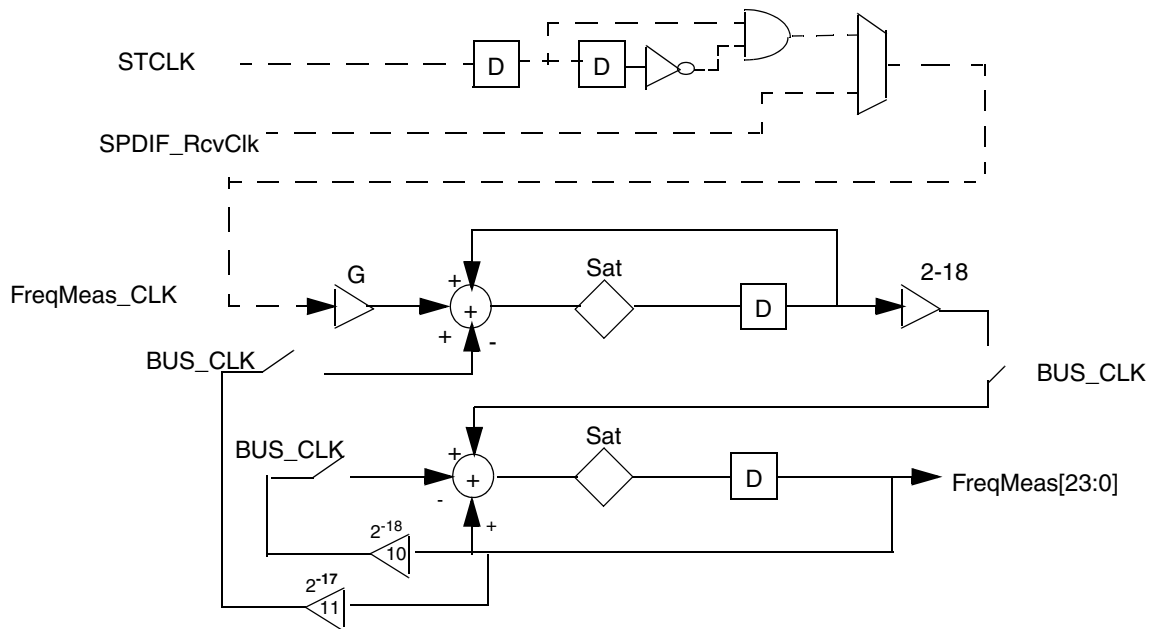


Figure 36-22. FreqMeas Circuit

All the D FlipFlop in the figure is clocked by BUS_CLK. The source clock to be measured is STCLK, which get through a synchronize logic and generate the FreqMeas_CLK. Associated with the measurement result, are 2 registers, PhaseConfig and FreqMeas. The circuit measures the frequency of the incoming clock as a function of the BUS_CLK. The circuit is a second-order filter. The output is a value represented by an unsigned number stored in the 24 bit FreqMeas register, giving the frequency of the source as a function of the BUS_CLK.

```
FreqMeas[23:0] = FreqMeas CLK/BUS CLK * 2**20 * COEF.
```

For example, if the COEF selection bits in PhaseConfig is written as 011, the COEF is now selected as 8, so the actual 'FreqMeas CLK/BUS CLK' is equal to 'FreqMeas[23:0]/2**23'.

As the internal bus is 32-bit, and the MSB is aligned, so the lower 8 bits ([24:31]) of Register FreqMeas always return zeros when FreqMeas is read by the processor.

36.4.2 SPDIF Transmitter

Audio data for the SPDIF transmitter is provided by the processor via EbuTxLeft and EbuTxRight registers. The channel status and user bits are also provided via corresponding registers. Clocking for the SPDIF transmitter is from input STCLKIN. The SPDIF transmitter clock source can be divided down as needed using TxClk_DF. The SPDIF transmitter output can be chosen from either the SPDIF transmitter block, directly from the SPDIF receiver via the output multiplexer or disabled.

The SPDIF transmitter generates a SPDIF output bitstream in IEC958 in the biphasic mark format. It consists of audio data, channel status and user bits.

36.4.2.1 Audio Data Transmission

Audio data for the SPDIF transmitter is provided by the processor via EbuTxLeft and EbuTxRight registers. They send audio data to Tx FIFOs, left and right. The Tx FIFOs are also 12-deep and 24-width (equal to the audio data width).

The audio data is transmitted as 24 bits, and the memory-mapped registers are 32 bits. When the audio data sent to the Tx FIFOs, MSB is aligned and the 24 MSBs ([0:23]) of EbuTxLeft and EbuTxRight are sent.

36.4.2.1.1 SPDIF Transmitter Data Registers—Behavior on Overrun, Underrun

The SPDIF Data Transmit registers (EbuTxLeft and EbuTxRight) have different FIFOs for left and right channel. As a result, there is always the possibility that left and right FIFOs may go out of sync due to FIFO underruns and FIFO overruns that affect only one part (left or right) of any FIFO. To prevent this from happening, hardware has been added on the device. Two mechanisms to prevent mismatch between the FIFOs are available.

If SPDIF Tx FIFO underruns on e.g. the right half of the FIFO, no sample leaves that FIFO. (because it was already empty.) Special hardware makes sure that the next sample read from the left FIFO does not leave the FIFO. (No read strobe is generated). If the underrun occurs on the left half of the FIFO, next read strobe to the right FIFO is blocked.

36.4.2.1.2 SPDIF Transmitter Data Registers—Automatic Resynchronization of FIFOs

See [Section 36.4.1.1, “Audio Data Reception.”](#)

36.4.2.1.3 EbuTxLeft, EbuTxRight Details

With SPDIF Tx FIFOs three exceptions are associated.

1. Empty
2. Under/overrun
3. Resync

When the empty condition (which is set by TxFIFOEmpty_Set field of the EBU_ConfigReg) is set for processor data output registers, the processor should write data to the FIFO, before underrun occurs. For example, when empty is set and 12 samples need to be written, it is acceptable for the software to write first 12 samples from the LEFT address, followed by 12 samples from the RIGHT address, or 1 sample

LEFT, followed by 1 sample RIGHT repeated 12 times. Left should be written before right. The implementation of all data out FIFOs is a double FIFO, one for left and one for right. Empty is set when both FIFOs are empty. Underrun, overrun are set when one of the FIFOs do underrun or do overrun. Resync is set when the hardware resynchronizes left and right FIFOs.

On receiving underrun, overrun interrupt, synchronization between Left and Right words in the FIFOs may be lost. Synchronization is not lost when the underrun or overrun comes from the IEC958 side of the FIFO. If the processor reads or writes more data from e.g. left than from right, synchronization is lost. If automatic resynchronization is enabled and the software obeys the rules to let this work, resynchronization is automatic.

36.4.2.2 Channel Status Transmission

A total of 48 Consumer channel status bits are transmitted from two registers. Channel Status Bits are ordered first bit left. CS-channel MSB bit 0 is located in bit position 0 in the memory-mapped register EBU_TxCChannelCons_h. CS-channel bit 23 is considered the LSB bit 23 in the register. C-channel bit 24 to 47 is seen as [0:23] bits of register EBU_TxCChannelCons_l. Because the two memory-mapped registers are 32 bits, 8 zeros are filled in bits field [24:31].

36.4.2.3 User Channel Transmission

The user channel transmitter is intended to assemble the CD subcode stream, conform the IEC908 CD standard specification. The generation of the data needs to be done by a software routine, and loaded into hardware registers. The device has provisions to insert this CD subcode stream into the outgoing IEC958 stream.

36.4.2.3.1 The CD-Text Format

CD User channel subcode may be transmitted by the IEC958 transmitter. This user channel subcode is assembled by application software inside the processor.

The CD-Text subcode has a 98-symbol packet structure. Of these 98 packets, the first 2 symbols are sync symbols, followed by 96 8-bit data symbols.

The boundaries of the 98-symbol packets are determined by free-run counters. The first two symbol of any packet are transmitted with the special sync sequence. The first and second symbols are all-0 symbols. The other 96 symbols need to be uploaded by the application software in register EBU_TxUChannel.

Upload is done by application software handshaking to interrupt UChannelTxEmpty. If this interrupt is set, the application software uploads 3 symbols of the current user channel packets into register EBU_TxUChannel.

The interrupt UChannelTxNextFirstByte flags the start of a new U channel packet. It is always coincident with UChannelTxEmpty, and signals that the first 3 symbols of a new packet need to be loaded into EBU_TxUChannel.

Pseudo-code to react on both interrupts. One interrupt handler can take care of both UchannelTxEmpty and UchannelTxNextFirstByte. This last interrupt is not enabled.

```

if(UchannelTxEmpty interrupt) then
if(UchannelTxNextFirstByte interrupt set also) then
    reset this interrupt
    synchronize pointer to sent out new frame
end if ;
load UchannelTransmit with data from pointer
update pointer
reset interrupt
end if ;

```

36.4.2.3.2 Advanced Topic: Synchronization of the Free-Running Counter—Legacy Only

For legacy reasons, it is possible to control which symbol is transmitted first on the EBU U-channel subcode interface.

To solve a legacy synchronization issue, the counter that determines the sync position is presettable via register CdTextControl.

36.4.2.3.3 Inserting CD User Channel Data into IEC958 Transmit Data

Source selection of data transmitted into the User Channel of IEC958 transmitter is selected by bits (22,23) of register EbuConfig.

- When selected source is off, zero is inserted in the user channel.
- When selected source is IEC958 receiver, every user channel data byte received into the input IEC958 user channel, is inserted into the outgoing stream at approx. the same time it was found in the incoming stream.
- When selected source is CD-Text, every data byte transmitted over the CD-Text output is also inserted into the IEC958 out stream. The most significant bit of every byte is transmitted as a 1. All sync symbols are transmitted as all-0.
- In case RCK clock is not present, it is possible to use the CD-Text interface to assemble the outgoing IEC958 User channel data. In this case, bit UChanTxTim in register CDText config must be set 1. It causes the timing to the CD-Text registers to be controlled by the IEC958 transmitter. One symbol (data or sync) is transmitted into the IEC958 output every 12 User Channel data bits.

36.4.2.4 Validity Flag Transmission

The validity bit setting is performed via bit 18 of the EBUConfig register.

Chapter 37

SRAM Memory (MEM)

37.1 Introduction

This chapter describes the multi-port on-chip static RAM (SRAM). The MEM contains 128 Kbytes of SRAM, divided into 4 banks of 32 Kbytes. The MEM block interface contains 3 bus ports. The MEM block is implemented as a cross-bar switch; all SRAM banks may service requests simultaneously to any of the master buses. Bus arbitration only occurs when two master bus ports try to access the same SRAM bank. The MEM block arbitrates between the bus ports with a fixed priority for each bus as follows: bus(1) (highest priority), bus2 (2nd priority), and bus(3) (lowest priority). The higher priority bus retains access to the SRAM as long as the bus maintains its request.

For the MPC5121e, the MEM ports are configured as follows:

- bus1 - connected to the CSB for PPC execution.
- bus2 - connected to the AXE.
- bus3 - connected to FEC, USB1, USB2, DMA2, SATA.

Figure 37-1 is a high-level block diagram of the memory block with its associated interfaces.

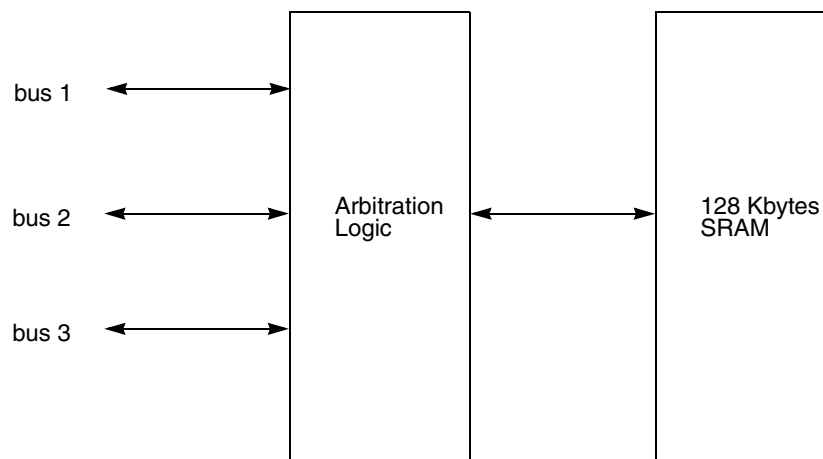


Figure 37-1. MEM Block Simplified Block Diagram

Chapter 38

Temperature Sensor

38.1 Introduction

The temperature sensor is an internal module to the MPC5121e that monitors the temperature of the chip, itself. There are two internal signals that the temperature sensor module asserts. The first signal is TEMP_105. The TEMP_105 signal asserts at one of eight programmable levels: 35°C, 45°C, 55°C, 65°C, 75°C, 85°C, 95°C, or 105°C. When the TEMP_105 signal asserts, an interrupt is signaled, if enabled, to the CPU core. The second signal, TEMP_125, is set to assert when the chip temperature reaches or exceeds 125°C. When the TEMP_125 signal asserts, a machine check exception is signaled to the CPU core. In either case, some type of machine check exception handler or interrupt service routine must be called that starts reducing the power consumption of the device, possibly by shutting down various modules or setting clocks to lower frequencies, etc.

The temperature measurement accuracy is limited to $\pm 5^{\circ}\text{C}$.

38.1.1 Normal Operation Mode

From the release of HRESET, the temperature sensor module is enabled and running. The TEMP_105 and TEMP_125 signals are active. There are several registers that control the state and condition of the temperature sensor module. These registers are located in modules other than the temperature sensor module and are described in detail there.

The temperature sensor module can be enabled or powered down by use of the TEMPPD Bit in the [Section 2.3.1.1.2, “System Priority Configuration Register \(SPCR\)”](#). Setting the TEMPPD bit to a logic 0 enables the temperature sensor module. Setting TEMPPD bit to a logic 1 puts the temperature sensor module in a power down condition. Powering down the temperature sensor module prevents it from asserting either the TEMP_105 or TEMP_125 signals.

The TEMP_105 can be programmed to one of eight temperature trip levels. The trip temperature is programmed using the TEMPSEL Bits in the System Priority Configuration Register.

The TEMP_125 signal is programmed to assert when the chip temperature reaches or exceeds 125°C. This temperature cannot be changed. When the TEMP_125 signal asserts, the TEMP125C bit sets in the [Section 20.2.1.15, “System Error Status Register \(SERSR\)”](#). If unmasked, the TEMP125C bit creates a machine check exception. The TEMP125C bit can be masked by the TEMP125C bit in the [Section 20.2.1.16, “System Error Mask Register \(SERMR\)”](#). The default state of the TEMP125C bit in the [Section 20.2.1.16, “System Error Mask Register \(SERMR\)”](#) is a logic 0 meaning that the TEMP_125C signal does not create a machine check exception. Setting the TEMP125C bit in the SERMR to a logic 1 allows the TEMP_125 signal to create a machine check exception.

The TEMP_105 signal is programmed to assert at one of eight programmable temperature levels. When the TEMP_105 signal asserts, the TEMP105C bit in the [Section 20.2.1.3, “System Internal Interrupt Pending Registers \(SIPNR_H and SIPNR_L\)”](#) asserts. If the TEMP105C bit in the SIPNR_L register asserts to a logic 1 and is not masked by the TEMP105C bit in the [Section 20.2.1.8, “System Internal Interrupt Mask Register \(SIMSR_H and SIMSR_L\)”](#) register, a normal interrupt is signaled to the CPU.

Chapter 39

Universal Serial Bus Interface with On-The-Go

39.1 Introduction

This chapter describes the universal serial bus (USB) interface of the MPC5121e. The USB interface implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications.

The following documents are available from the USB Implementers Forum web page at:

<http://www.usb.org/developers/docs/>

- *Universal Serial Bus Specification*, Rev. 2.0
- *On-The-Go Supplement to the USB 2.0 Specification*, Rev. 1.0a

The following documents are available from the Intel USB Specifications web page at:

<http://www.usb.org/developers/docs/>

- *Enhanced Host Controller Interface Specification for Universal Serial Bus*, Rev. 1.0
- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification*, Ver. 1.05

The following documents are available from the ULPI web page at:

<http://www.usb.org/developers/docs/>

- *UTMI+ Specification*, Rev. 1.0
- *UTMI Low Pin-Count Interface (ULPI) Specification*, Rev. 1.0

39.1.1 Overview

The MPC5121e implements two USB modules, having a dual-role (DR) or On-The-Go (OTG) capabilities. The USB0 and USB1 can be connected to an external PHY using the ULPI protocol. In addition, the USB0 can be connected to an on-chip UTMI+ PHY. Collectively, the modules and external ports are called the USB interface.

39.1.2 Features

The USB0 and USB1 modules supplies the following features:

- Supports USB device mode
- Supports USB On-The-Go mode including host capability
- Complies with USB specification Rev 2.0
- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbit/s) operations
- Supports external PHY with UTMI+ low pin count (ULPI) interface

- Supports operation as a standalone USB device
 - Supports one upstream facing port
 - Supports four programmable, bidirectional USB endpoints
- Host Mode supports direct connect of LS/FS devices

In addition, the USB0 module supports an internal PHY with UTMI+ interface.

39.1.3 Modes of Operation

The USB0 /USB1 has three basic operating modes: Host, Device, and On-the-Go (OTG).

The USB0 module uses default on-chip PHY (UTMI+ interface). In addition to the on-chip PHY, the USB0 can use an external ULPI PHY.

The USB1 can work only with an external ULPI PHY.

39.2 Memory Map/Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers. [Table 39-1](#) shows the memory map of the USB interface.

The USB1 has a base address of 0x00_3000; the USB0 has a base address of 0x00_4000.

Table 39-1. USB Interface Memory Map

Offset	Register	Access	Section/Page
USB0 /USB1 Controller Registers			
0x000	ID—Identification register	R	39.2.1.1/3939-7
0x004	HWGENERAL—General hardware parameters	R	39.2.1.1/3939-8
0x008	HWHOST—Host hardware parameters	R	39.2.1.1/3939-9
0x00C	HWDEVICE—Device hardware parameters	R	39.2.1.1/3939-10
0x010	HWTXBUF—TX buffer hardware parameters	R	39.2.1.1/3939-11
0x014	HWRXBUF—RX buffer hardware parameters	R	39.2.1.1/3939-12
0x080	GPTIMER0LD—General Purpose Timer load value	R/W	39.2.1.3/3939-19
0x084	GPTIMER0CTRL—General Purpose Timer 0 Control	R/W	39.2.1.3/3939-19
0x088	GPTIMER1LD—General Purpose Timer load value	R/W	39.2.1.3/3939-19
0x08C	GPTIMER1CTRL—General Purpose Timer 0 Control	R/W	39.2.1.3/3939-19
0x090	SBUSCFG—System Bus Interface Control	R/W	39.2.1.1/3939-12
0x100	CAPLENGTH—Capability register length	R	39.2.1.2/3939-14
0x102	HCVERSION—Host interface version number	R	39.2.1.2/3939-14
0x104	HCCPARAMS—Host ctrl. structural parameters	R	39.2.1.2/3939-15
0x108	HCCPARAMS—Host ctrl. capability parameters	R	39.2.1.2/3939-16
0x120	DCVERSION—Device interface version number	R	39.2.1.2/3939-17

Table 39-1. USB Interface Memory Map (continued)

Offset	Register	Access	Section/Page
0x124	DCCPARAMS—Device controller parameters	R	39.2.1.2/3939-18
0x140	USBCMD—USB command	R/W	39.2.1.4/3939-21
0x144	USBSTS—USB status	R/W	39.2.1.4/3939-24
0x148	USBINTR—USB interrupt enable	R/W	39.2.1.4/3939-27
0x14C	FRINDEX—USB frame index	R/W	39.2.1.4/3939-29
0x154	PERIODICLISTBASE—Frame list base address	R/W	39.2.1.4/3939-31
0x154	DEVICEADDR—USB device address	R/W	39.2.1.4/3939-32
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode)	R/W	39.2.1.4/3939-33
0x158	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	39.2.1.4/3939-34
0x15C	TTCTRL— TT status and control	R/W	39.2.1.4/3939-35
0x160	BURSTSIZE—Programmable burst size	R/W	39.2.1.4/3939-36
0x164	TX TTFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	39.2.1.4/3939-37
0x170	ULPI Viewport	R/W	39.2.1.4/3939-39
0x178	ENDPTNAK—End Point NAK	R/W	39.2.1.4/3939-41
0x17C	ENDPTNAKEN—End Point NAK Enable	R/W	39.2.1.4/3939-42
0x180	CONFIGFLAG—Configured flag register	R	39.2.1.4/3939-43
0x184	PORTSC1—Port status/control	R/W	39.2.1.4/3939-43
0x1A4	OTGSC—On-the-Go status and control	R/W	39.2.1.4/3939-48
0x1A8	USBMODE—USB device mode	R/W	39.2.1.4/3939-51
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	39.2.1.4/3939-54
0x1B0	ENDPTPRIME—Endpoint initialization	R/W	39.2.1.4/3939-55
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	39.2.1.4/3939-56
0x1B8	ENDPTSTATUS—Endpoint status	R	39.2.1.4/3939-57
0x1BC	ENDPTCOMPLETE—Endpoint complete	R/W	39.2.1.4/3939-58
0x01C0	ENDPTCTRL0—Endpoint control 0	R/W	39.2.1.4/3939-59
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	39.2.1.4/3939-61
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	39.2.1.4/3939-61
0x1CC	ENDPTCTRL3—Endpoint control 3	R/W	39.2.1.4/3939-61
0x200	USBGENCTRL — USB General Control	R/W	39.2.1.4/3939-63
0x204	ISIPHYCTRL — On-Chip PHY Control	R/W	39.2.1.4/3939-64
0x208– 0xFFFF	Reserved	—	—

The following sections provide details about the registers in the USB memory map.

Power Architecture		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
Addr	Register																																			
000	ID	Reserved								Revision								1	1	NID						0	0	ID								
004	HWGENE RAL	Reserved																						S M	PHYM			PHYW			B W T	CLKC		RT		
008	HWHOST	TTPER								TTASY								Reserved										NPORT				H C				
00C	HWDEVIC E	Reserved																								DEVEP				D C						
010	HWTXB UF	TX CL R	Reserved								TXCHANADD								TXADD						TXBURST											
014	HWRXB UF	Reserved																RXADD						RXBURST												
080	GPTIMER OLD	Reserved								GPTLD																										
084	GPTIMER OCTRL	G P T R U N	G P T R S T	Reserved					G P T M O D E	GPTCNT																										
088	GPTIMER 1LD	Reserved								GPTLD																										
08C	GPTIMER 1CTRL	G P T R U N	G P T R S T	Reserved					G P T M O D E	GPTCNT																										
100	CAPLENG TH	HCIVERSION																Reserved						CAPLENGTH												
102	HCIVERSI ON	HCIVERSION																HCIVERSION																		
104	HCCPARA MS	Reserved				N_TT				N_PPT				Reserved		PI	N_CC				N_PPC				Reserved		PP C	N_PORTS								
108	HCCPARA MS	Reserved																EECP						IST						AS P	PF L	AD C				
120	DCIVERSI ON	Reserved																DCIVERSION																		
124	DCCPARA MS	Reserved																						H C	D C	Reserv ed		DEN								
140	USBCMD	Reserved								ICT						FS 2	AT DT W	SU T W	R	AS PE			ASP		LR	IA A	AS E	PS E	FS	RS T	RS					
144	USBSTS	Reserved						TI 1	TI 0	Reserved				UP I	UA I			NA KI	AS	PS	R CL	H CH			UL PII			SL I	SR I	U RI	AA I	SE I	FR I	PC I	UE I	UI
148	USBINTR	Reserved						TI E1	TI E0	Reserved				UP IA	UA IE			NA KE	Reserved						UL PI E			SL E	SR E	U RE	AA E	SE E	FR E	PC E	UE E	UE
14C	FRINDEX	Reserved																		FRINDEX																
154	PERIODIC LISTBASE	PERBASE																				Reserved														

Figure 39-1. Summary of Register Layout

Power Architecture		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
154	DEVICEA DDR	USBADR							US BA D RA	Reserved																							
158	ASYNCLI STADDR	ASYBASE																												Reserved			
158	ENDPOIN TLISTADD R	EPBASE																				Reserved											
15C	TTCTRL	R	TTHA							Reserved																							

Figure 39-1. Summary of Register Layout (continued)

Power Architecture		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																		
Conventional		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
160	BURSTSIEZE	Reserved															TXPBURST							RXPBURST																											
164	TXFILLTURNING	Reserved										TXFIFOTHRES					Reserved			TXSCHEALTH					TXSCHOH																										
170	ULPIVIEWPORT	ULPIWU	ULPIRU	ULPIRW	R	ULPISS	ULPIPORT			ULPIADDR					ULPIDATARD							ULPIDATAWR																													
178	ENDPTNAK												EPTN																	EPRN																					
17C	ENDPTNAKEN												EPTNE																	EPRNE																					
180	CONFIGFLAG	Reserved																														1																			
184	PORTSC1	PTS		STS		PTW		PSPD			PFSC		PHCD		WKOC		WKDC		WKC		PTC			PIC		PO		PP		LS		HSP		PR		SUP		FPR		OCC		OCA		PEC		PE		CSC		CCS	
1A4	OTGSC		DPIE	1msE	BSIE	BSVE	ASVE	AVVE	IDIE		DPIS	1mss	BSIS	BSVIS	ASVIS	AVVIS	IDIS		DPS	1mst	BSE	BSV	ASV	AVV	ID	HABA	HADP	IDPU	DP	OT	HAR	VC	VD																		
1A8	USBMODE	Reserved																								VPBS		SDIS		SLOM		ES		CM																	
1AC	ENDPTSETUPSTAT	Reserved															ENDPTSETUPSTAT																																		
1B0	ENDPTPRIME	Reserved											PETB				Reserved											PERB																							
1B4	ENDPTFLUSH	Reserved											FETB				Reserved											FERB																							
1B8	ENDPTSTATUS	Reserved											ETBR				Reserved											ERBR																							
1BC	ENDPTCOMPLETE	Reserved											ETCE				Reserved											ERCE																							
1C0	ENDPTCTRL0	Reserved								TXE		Reserved			TXT			TXS		Reserved							RXE		Reserved			RXT			RXS																
1C4	ENDPTCTRL1	Reserved								TXE		TXR		TXI			TXT		TXD		TXS		Reserved							RXE		RXR		RXI			RXT		RXD		RXS										
1C8	ENDPTCTRL2	Reserved								TXE		TXR		TXI			TXT		TXD		TXS		Reserved							RXE		RXR		RXI			RXT		RXD		RXS										
1CC	ENDPTCTRL3	Reserved								TXE		TXR		TXI			TXT		TXD		TXS		Reserved							RXE		RXR		RXI			RXT		RXD		RXS										
200	USBGENCTRL	Reserved																										ULPISEL		PPP		PPP		WULPIEN		WUIE															
204	ISIPHYCTRL	Reserved																										OCO		BSBH		BSBH		LSFE		PXFE															

Figure 39-1. Summary of Register Layout (continued)

39.2.1 Register Descriptions

39.2.1.1 Module Identification Registers

The identification registers declare the slave interface presence and include hardware configuration parameters. These registers include the identification register, the general hardware parameters register, the host hardware parameters register, the device hardware parameters register, the Tx buffer hardware parameters register, and the Rx buffer hardware parameter register. These are not defined by the EHCI specification.

39.2.1.1.1 Identification (ID) Register (Non-EHCI)

The ID register provides a simple way to determine if the module is present in the system. The ID register identifies the module and its revision.

Figure 39-2 shows the ID register. Table 39-2 provides bit descriptions for the ID register.

Offset: Base + 000h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	REVISION							
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	1	1	NID					0	0	ID						
W																
Reset	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1

= Unimplemented or Reserved

Figure 39-2. Identification (ID) Register

Table 39-2. ID Register Field Descriptions

Field	Description
REVISION	Revision number of the module.
NID	Ones complement version of ID[5:0].
ID	Configuration number. This number is set to 0x05.

39.2.1.1.2 General Hardware Parameters (HWGENERAL) Register (non-EHCI)

The HWGENERAL register contains parameters that define the particular implementation of the module. Figure 39-3 shows the HWGENERAL register. Table 39-3 provides bit descriptions for the HWGENERAL register.

Offset: Base + 004h, (USB0)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	SM		PHYM			PHYW		BWT	CLKC		RT
W																
Reset	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	1

Offset: Base + 004h, (USB1)

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	SM		PHYM			PHYW		BWT	CLKC		RT
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1

□ = Unimplemented or Reserved

Figure 39-3. General Hardware Parameters (HWGENERAL) Register

Table 39-3. HWGENERAL Register Field Descriptions

Field	Description
SM	SERIAL_MODE. Always 0 indicating no serial interface engine is present.
PHYM	PHY_MODE. For the USB0, these bits reads out 100, indicating the USB transceiver is under software control and resets to UTMI+. For the USB1, these bits reads out 010, indicating the USB transceiver can only work with the ULPI interface.
PHYW	PHY Width. This field is only relevant for UTMI mode, therefore it is only relevant to the USB0 module in UTMI mode. For the USB0 module, PHYW always reads 11, indicating the UTMI interface width is programmable and defaults to 16-bits wide.

BWT	Reserved for manufacturing test. Reads as 0.
CLKC	Reserved for manufacturing test. Indicates the system and PHY clock partitioning of the design. Default 10.
RT	Reserved for manufacturing test. Always 1.

39.2.1.1.3 Host Hardware Parameters (HWHOST) Register (non-EHCI)

The HWHOST register provides host hardware parameters for this implementation of the module.

Figure 39-4 shows the HWHOST register. Table 39-4 provides bit descriptions for the HWHOST register.

Offset: Base + 008h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TTPER								TTASY							
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	NPORT			HC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 39-4. Host Hardware Parameters (HWHOST) Register

Table 39-4. HWHOST Register Field Descriptions

Field	Description
TTPER	Transaction translator periodic contexts. The number of supported transaction translator periodic contexts. This always reads 0x10 (d'16).
TTASY	Transaction translator contexts. The number of transaction translator contexts.
NPORT	Always 0, indicating the number of ports available (NPORT + 1) for this host implementation.
HC	Always 1, indicating the module is host capable.

39.2.1.1.4 Device Hardware Parameters (HWDEVICE) Register (Non-EHCI)

This register is not defined in the EHCI specification. The HWDEVICE register provides device hardware parameters for this implementation of the OTG module. [Figure 39-5](#) shows the HWDEVICE register.

[Table 39-5](#) provides bit descriptions for the HWDEVICE register.

Offset: Base + 00Ch

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	DEVEP					DC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1


 = Unimplemented or Reserved

Figure 39-5. Device Hardware Parameters (HWDEVICE) Register

Table 39-5. HWDEVICE Register Field Descriptions

Field	Description
DEVEP	Device endpoints. The number of supported endpoints; always 0x4.
DC	Always 1 indicating the OTG module is device capable.

39.2.1.1.5 Transmit Buffer Hardware Parameters (HWTXBUF) Register (non-EHCI)

The HWTXBUF register provides the transmit buffer parameters for this implementation of the module. Figure 39-6 shows the HWTXBUF register. Table 39-6 provides bit descriptions for the HWTXBUF register.

Offset: Base + 010h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TXLCR	0	0	0	0	0	0	0	TXCHANADD							
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXADD								TXBURST							
W																
Reset	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 39-6. Transmit Buffer Hardware Parameters (HWTXBUF) Register

Table 39-6. HWTXBUF Register Field Descriptions

Field	Description
TXLCR	Reserved. Always 1.
TXCHANADD	Transmit channel address. The number of address bits required to address one channel's worth of TX data. Always 0x7 (7).
TXADD	Transmit address. The number of address bits for the entire TX buffer. Always 0x9 (9).
TXBURST	Transmit burst. Indicates the number of data beats in a burst for transmit DMA data transfers. Always 0x8 (8).

39.2.1.1.6 Receive Buffer Hardware Parameters (HWRXBUF) Register (non-EHCI)

The HWRXBUF register provides the receive buffer parameters for this implementation of the module. [Figure 39-7](#) shows the HWRXBUF register. [Table 39-7](#) provides bit descriptions for the HWRXBUF register.

Offset: Base + 014h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RXADD								RXBURST							
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 39-7. Receive Buffer Hardware Parameters (HWRXBUF) Register

Table 39-7. HWRXBUF Register Field Descriptions

Field	Description
RXADD	Receive address. The number of address bits for the entire RX buffer. Always 0x8 (8).
RXBURST	Receive burst. Indicates the number of data beats in a burst for receive DMA data transfers. Always 0x8 (8).

39.2.1.1.7 System Bus Interface Configuration (SBUSCFG) (Non-EHCI)

The SBUSCFG register contains the control for the systembus interface. [Figure 39-8](#) shows the SBUSCFG register. [Table 39-8](#) provides bit descriptions for the SBUSCFG register.

Offset: Base + 090h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	AHBBRST		
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0


 = Unimplemented or Reserved

Figure 39-8. System Bus Interface Configuration (SBUSCFG) Register

Table 39-8. SBUSCFG Register Field Descriptions

Field	Description
AHBBRST	<p>Selects the following options for the m_nburst signal of the AMBA Master Interface</p> <p>000: INCR burst of unspecified length 001: INCR4, non-multiple transfers of INCR4 is decomposed into singles 010: INCR8, non-multiple transfers of INCR8, is decomposed into INCR4 or singles 011: INCR16, non-multiple transfers of INCR16, is decomposed into INCR8, INCR4 or singles 100: This value is reserved and should not be used 101: INCR4, non-multiple transfers of INCR4 is decomposed into smaller unspecified length bursts 110: INCR8, non-multiple transfers of INCR8 is decomposed into smaller unspecified length bursts 111: INCR16, non-multiple transfers of INCR16 is decomposed into smaller unspecified length bursts</p> <p>In all cases where the unspecified length burst is allowed, singles access may also occur, this is mostly true when the transaction is not 32-bit aligned.</p> <p>Two consecutive single accesses should not happen.</p> <p>When a INCRx burst size is selected and the transfer is not multiple of the INCRx burst, the burst is decomposed in the different ways. With AHBBRST[2] = 1, the smaller bursts is unspecified length. with AHBBRST[2] = 0, the smaller bursts are smaller INCRx or singles. For example, lets say that it's required at a given time, to transfer 22 words of information, for the following values of AHBBRST the master sequence are:</p> <p>101: INCR4+ INCR4 +INCR4+ INCR4 +INCR4+ INCR unspec. length 110: INCR8+INCR8+INCR4+ INCR unspec. length 111: INCR16+INCR4+ INCR unspec. length 001: INCR4+ INCR4 +INCR4+ INCR4 +INCR4+SINGLE+SINGLE 010: INCR8+INCR8+INCR4+SINGLE+SINGLE 011: INCR16+INCR4+SINGLE+SINGLE</p> <p>When this field is different from zero, the value of the fields TXBURST/RXBURST in register BURSTSIZE 160h, is ignored by the controller.</p> <p>Internally the BURSTSIZE is set to the value of the INCRx AMBA burst. Since this has a direct relation with the burst sizes you must be careful with AHB burst selected. Although the TXBURST/RXBURST are bypassed, this register can be written/read with no effect while the AHBBRST field is non-zero.</p> <p>Note: Setting the AHBBRST value to 000 might cause bus allocation during BULK or ISO transfers.</p> <p>Note: Changing this AHBBRST field while an AMBA AHB transaction is in progress yields undefined results! One possible way of ensuring that no undefined results occur is to set the runstop bit to zero in the USB_CMD register, after the HCHALTED is detected in USBSTS.</p>

39.2.1.2 Capability Registers

The capability registers specify the software limits, restrictions, and capabilities of the host/device controller implementation. The EHCI specification defines most of these registers. Registers not defined by the EHCI specification are noted in their descriptions.

39.2.1.2.1 Capability Registers Length (CAPLENGTH)

This register is an offset to add to the register base address to find the beginning of the operational register space, the location of the USBCMD register. [Figure 39-9](#) shows the CAPLENGTH register. [Table 39-9](#) provides bit descriptions for the CAPLENGTH register.

Offset: Base + 100h

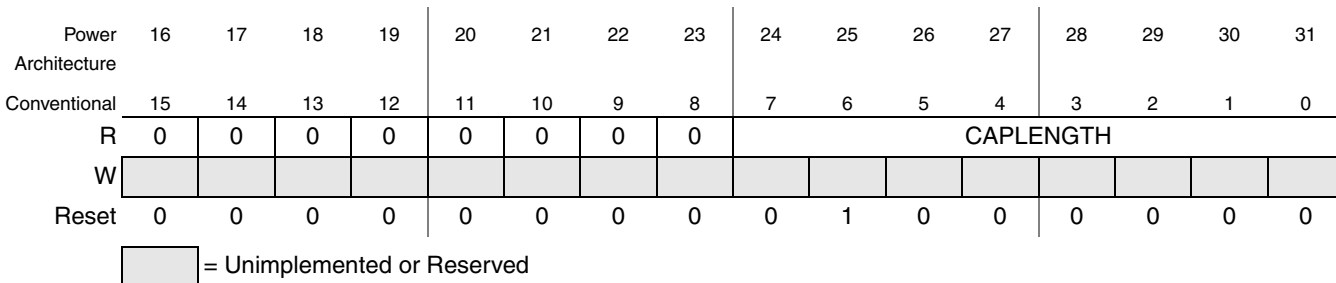


Figure 39-9. Capability Registers Length (CAPLENGTH)

Table 39-9. CAPLENGTH Register Field Descriptions

Field	Description
CAPLENGTH	Capability registers length. Value is 0x40.

39.2.1.2.2 Host Controller Interface Version (HCVERSION)

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this host controller. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. [Figure 39-11](#) shows the HCVERSION register. [Table 39-10](#) provides bit descriptions for the HCVERSION register.

Offset: Base + 102h

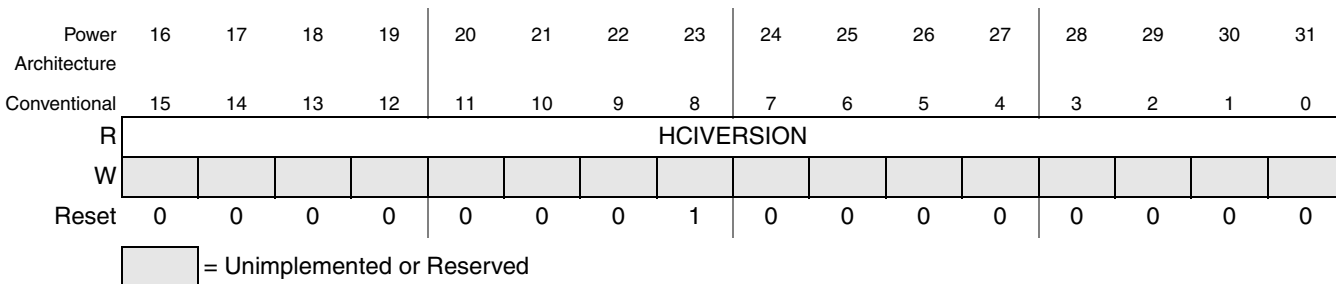


Figure 39-10. Host Controller Interface Version (HCVERSION) Register

Table 39-10. HCVERSION Register Field Descriptions

Field	Description
-------	-------------

HCVERSION	EHCI revision number. Value is 0x0100 indicating version 1.0.
-----------	---

39.2.1.2.3 Host Controller Structural Parameters (HCSPARAMS) (EHCI-Compliant)

This register contains structural parameters such as the number of downstream ports. [Figure 39-11](#) shows the HCSPARAMS register. [Table 39-11](#) provides bit descriptions for the HCSPARAMS register.

Offset: Base + 104h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	N_TT				N_PTT				0	0	0	PI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	N_CC				N_PCC				0	0	0	PPC	N_PORTS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1


 = Unimplemented or Reserved

Figure 39-11. Host Controller Structural Parameters (HCSPARAMS) Register

Table 39-11. HCSPARAMS Register Field Descriptions

Field	Description
N_TT	Number of transaction translators. This field is always reads 0.
N_PTT	Ports per transaction translator. This field is always reads 0.
PI	Port indicators. This bit indicates whether the ports support port indicator control. Always 1. 1 The port status and control registers include a r/w field for controlling the state of the port indicator.
N_CC	Number of Companion Controllers. This field indicates the number of companion controllers associated with the OTG controller. This field is always 0.
N_PCC	Number of Ports per Companion Controller. This field is always 0.
PPC	Power Port Control. This bit indicates whether the host controller supports port power control.
N_PORTS	Number of Ports. This field indicates the number of physical downstream ports implemented for host applications. The value of this field determines how many port registers are addressable in the operational register.

39.2.1.2.4 Host Controller Capability Parameters (HCCPARAMS)

This register identifies multiple mode control (time-base bit functionality) addressing capability. Figure 39-12 shows the HCCPARAMS register. Table 39-12 provides bit descriptions for the HCCPARAMS register.

Offset: Base + 108h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EECP								IST				0	ASP	PFL	ADC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

= Unimplemented or Reserved

Figure 39-12. Host Controller Capability Parameters (HCCPARAMS)

Table 39-12. HCCPARAMS Register Field Descriptions

Field	Description
EECP	EHCI Extended Capabilities Pointer. This optional field indicates existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device. This field is always 0.
IST	Isochronous Scheduling Threshold. This field indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit [7] is 0, the value of the least significant three bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit [7] is a 1, host software assumes the host controller may cache an isochronous data structure for an entire frame. This field is always 0.
ASP	Asynchronous Schedule Park Capability. This bit indicates if the host controller supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This field is always 1 (park feature supported).
PFL	Programmable Frame List Flag. This bit indicates system software can specify and use a frame list length less than 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K page boundary. This requirement ensures the frame list is always physically contiguous. This field is always 1.
ADC	64-Bit Addressing Capability. This field is always 0; Only 32-bit addressing is supported. 0 Data structures use 32-bit address memory pointers.

39.2.1.2.5 Device Controller Interface Version (DCIVERSION)—Non-EHCI

This register is not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. [Figure 39-13](#) shows the DCIVERSION register. [Table 39-13](#) provides bit descriptions for the DCIVERSION register.

Offset: Base + 120h

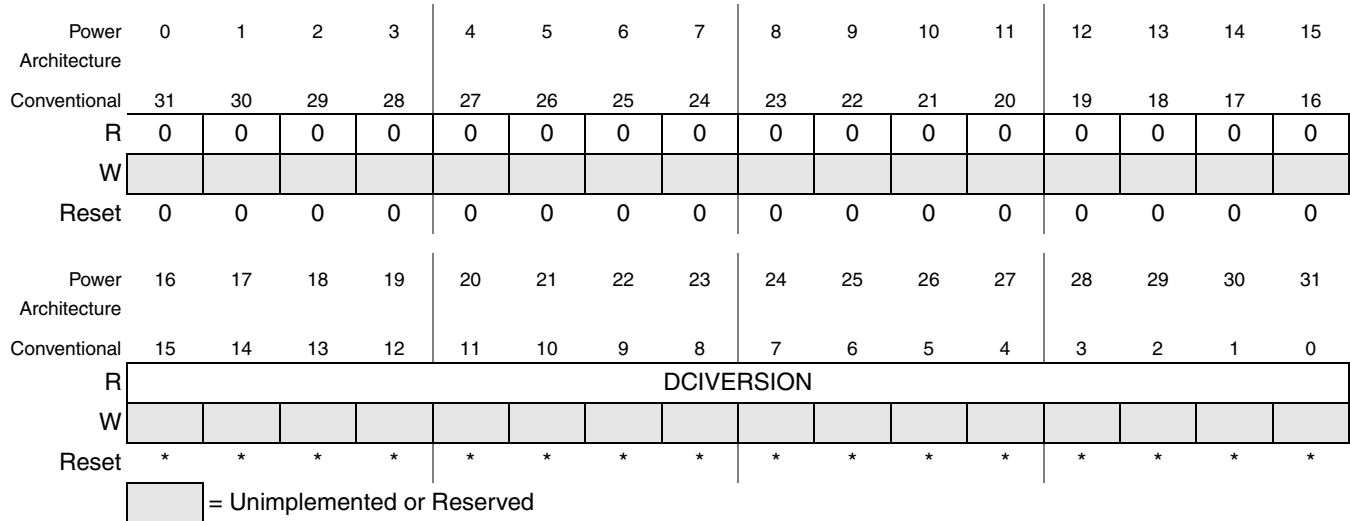


Figure 39-13. Host Controller Interface Version (DCIVERSION) Register

Table 39-13. HCIVERSION Register Field Descriptions

Field	Description
DCIVERSION	Device interface revision number: * is implementation dependent. The device controller interface conforms to the two-byte BCD encoding of the interface version number contained in this register.

39.2.1.2.6 Device Controller Capability Parameters (DCCPARAMS)—Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the OTG module. Figure 39-14 shows the DCCPARAMS register. Table 39-14 provides bit descriptions for the DCCPARAMS register.

Offset: Base + 124h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	HC	DC	0	0	DEN				
W																
Reset	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0


 = Unimplemented or Reserved

Figure 39-14. Device Controller Capability Parameters (DCCPARAMS)

Table 39-14. DCCPARAMS Register Field Descriptions

Field	Description
HC	Host Capable. Always 1 indicating the OTG controller can operate as an EHCI compatible USB 2.0 host
DC	Device Capable. Always 1, indicating the OTG controller can operate as an USB 2.0 device. 0 No device capability (host only). 1 Device capability.
DEN	Device Endpoint Number. This field indicates the number of endpoints built into the device controller. Always 0x4.

39.2.1.3 Device/Host Timer Registers (Non-EHCI)

The host/device Controller drivers can measure time related activities using these timer registers. These registers are not part of the standard EHCI controller.

There are two timers available (GPTIMER0LD and GPTIMER1LD), each with its own control register (GPTIMER0CTRL and GPTIMER1CTRL).

Figure 39-15 shows the GPTIMERxLD register. Table 39-15 provides bit descriptions for the GPTIMERxLD register.

Figure 39-16 shows the GPTIMERxCTRL register. Table 39-16 provides bit descriptions for the GPTIMERxCTRL register.

Offset:

GPTIMER0LD: Base + 80h

GPTIMER1LD: Base + 88h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	GPTLD							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	GPTLD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-15. General Purpose Timer Load Register (GPTIMERxLD)

Table 39-15. GPTIMERxLD Register Field Descriptions

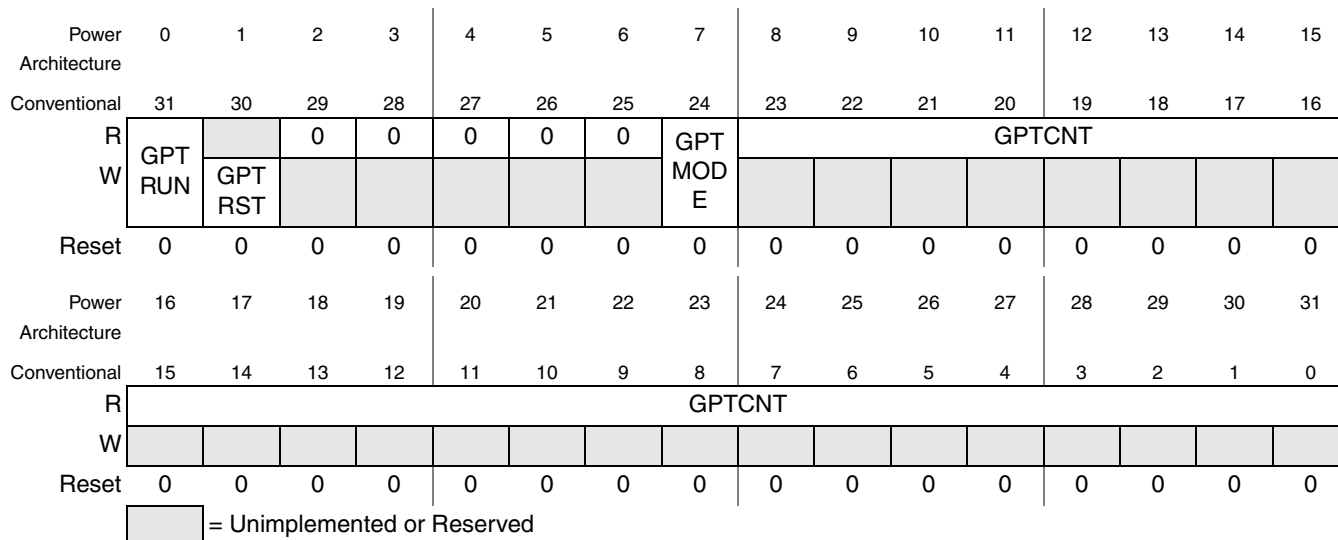
Field	Description
GPTLD	GPTLD--General Purpose Timer Load Value. This field is the value loaded into the GPTCNT countdown timer on a reset action. This value represents the time in microseconds minus 1 for the timer duration. Note: The maximum value is 0xFFFFF.

This register contains the control for each timer and a data field can be queried to determine the running count value. The timer has a granularity of 1 us and can be programmed to a little over 16 seconds. There are two modes supported by the timer; the first is a one-shot, and the second is a looped count described in Table 39-17. When the timer counter value transitions to 0, an interrupt can be generated through the use of the timer interrupts in the USBSTS and USBINTR registers.

Offset:

GPTIMER0CTRL: Base + 84h

GPTIMER1CTRL: Base + 8Ch

**Figure 39-16. General Purpose Timer Control Register (GPTIMERxCTRL)****Table 39-16. GPTIMERxCTRL Register Field Descriptions**

Field	Description
GPTRUN	GPTRUN – General Purpose Timer Run. This bit enables the general purpose timer to run. Setting or clearing this bit does not have an effect on the GPTCNT value. 0 Timer Stop; 1 Timer Run
GPTRST	GPTRST – General Purpose Timer Reset. Writing a 1 to this bit reloads the GPTCNT with the value in GPTLD. 0 No action 1 Load counter
GPTMODE	GPTMODE – General Purpose Timer Mode. This bit selects between a single timer countdown and a looped count down. In one-shot mode, the timer counts down to zero, generate an interrupt and stop until the counter is reset by software. In repeat mode, the timer counts down to zero, generate an interrupt, and automatically reload the counter to begin again.
GPTCNT	GPTCNT – General Purpose Timer Counter. Contains the value of the running timer.

39.2.1.4 Operational Registers

The operational registers are comprised of dynamic control or status registers that may be read-only, read/write, or read/write-1-to-clear. The following sections define the operational registers.

39.2.1.4.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Offset: Base + 140h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	ITC							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	0	ASPE	0	ASP		LR	IAA	ASE	PSE	FS1	FS0	RST	RS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-17. USB Command Register (USBCMD)

Table 39-17. USBCMD Register Field Descriptions

Field	Description
ITC	Interrupt Threshold Control. The system software uses this field to set the maximum rate at which the module issues interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 40 microframes
FS2	See bit 3:2 below. This is a non-EHCI bit.
ATDTW	Add dTD TripWire. This is a non-EHCI bit is present on the OTG module only. This bit acts as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit is also cleared by hardware when the state machine is in a hazard region where adding a dTD to a primed endpoint may go unrecognized. More information on the use of this bit is described in Section 39.8.10.1, “Device Operation,” of this manual.
SUTW	Setup TripWire. This is a non-EHCI bit present on the OTG module only. This bit acts as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE), a hazard exists when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by software or cleared by hardware. More information on the use of this bit is described in Section 39.8.10.1, “Device Operation,” of this manual.
ASPE	Asynchronous Schedule Park Mode Enable. Software uses this bit to enable or disable Park mode. 1 Enabled. 0 Disabled.

Table 39-17. USBCMD Register Field Descriptions (continued)

Field	Description																																				
ASP	Asynchronous Schedule Park Mode Count. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the asynchronous schedule before continuing traversal of the asynchronous schedule. Valid values are 0x1 to 0x3. Software must not write a 0 to this field when park mode enable is a 1 as this results in undefined behavior.																																				
LR	Light Host/Device Controller Reset (OPTIONAL). Not Implemented. Always 0.																																				
IAA	<p>Interrupt on Async Advance Doorbell. Software uses this bit as a doorbell to tell the controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell.</p> <p>When the controller has evicted all appropriate cached schedule states, it sets the interrupt on async advance status bit in the USBSTS register. If the interrupt on sync advance enable bit in the USBINTR register is 1, the host controller asserts an interrupt at the next interrupt threshold.</p> <p>The controller sets this bit to 0 after it has set the interrupt on sync advance status bit in the USBSTS register to 1. Software should not write a 1 to this bit when the asynchronous schedule is inactive. Doing so yields undefined results.</p> <p>This bit is only used in host mode. Writing a 1 to this bit when the OTG module is in device mode has undefined results.</p>																																				
ASE	<p>Asynchronous Schedule Enable. This bit controls whether the controller skips processing the asynchronous schedule. Only used in host mode.</p> <p>1 Use the ASYNCLISTADDR register to access the asynchronous schedule.</p> <p>0 Do not process the asynchronous schedule.</p>																																				
PSE	<p>Periodic Schedule Enable. This bit controls whether the controller skips processing the periodic schedule. Only used in host mode.</p> <p>1 Use the PERIODICLISTBASE register to access the periodic schedule.</p> <p>0 Do not process the periodic schedule.</p>																																				
FS[1:0]	<p>Frame List Size. With bit 15, these bits make the FS[2:0] field. This field is read/write only if programmable frame list flag in the HCCPARAMS registers is set to 1. This field specifies the size of the frame list that controls which bits in the frame index register should be used for the frame list current index. Used only in host mode. Values below 256 elements are not defined in the EHCI specification.</p> <table><thead><tr><th>FS2</th><th>FS1</th><th>FS0</th><th>Descriptions</th></tr></thead><tbody><tr><td>0</td><td>0</td><td>0</td><td>1024 elements (4096 bytes)</td></tr><tr><td>0</td><td>0</td><td>1</td><td>512 elements (2048 bytes)</td></tr><tr><td>0</td><td>1</td><td>0</td><td>256 elements (1024 bytes)</td></tr><tr><td>0</td><td>1</td><td>1</td><td>128 elements (512 bytes)</td></tr><tr><td>1</td><td>0</td><td>0</td><td>64 elements (256 bytes)</td></tr><tr><td>1</td><td>0</td><td>1</td><td>32 elements (128 bytes)</td></tr><tr><td>1</td><td>1</td><td>0</td><td>16 elements (64 bytes)</td></tr><tr><td>1</td><td>1</td><td>1</td><td>8 elements (32 bytes)</td></tr></tbody></table>	FS2	FS1	FS0	Descriptions	0	0	0	1024 elements (4096 bytes)	0	0	1	512 elements (2048 bytes)	0	1	0	256 elements (1024 bytes)	0	1	1	128 elements (512 bytes)	1	0	0	64 elements (256 bytes)	1	0	1	32 elements (128 bytes)	1	1	0	16 elements (64 bytes)	1	1	1	8 elements (32 bytes)
FS2	FS1	FS0	Descriptions																																		
0	0	0	1024 elements (4096 bytes)																																		
0	0	1	512 elements (2048 bytes)																																		
0	1	0	256 elements (1024 bytes)																																		
0	1	1	128 elements (512 bytes)																																		
1	0	0	64 elements (256 bytes)																																		
1	0	1	32 elements (128 bytes)																																		
1	1	0	16 elements (64 bytes)																																		
1	1	1	8 elements (32 bytes)																																		

Table 39-17. USBCMD Register Field Descriptions (continued)

Field	Description
RST	<p>Controller Reset. Software uses this bit to reset the controller. This bit is set to 0 by the controller when the reset process is complete. Software cannot terminate the reset process early by writing a 0 to this register.</p> <p>Host Mode:</p> <p>When software writes a 1 to this bit, the Host Controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit to a 1 when the HCHalted bit in the USBSTS register is a zero. Attempting to reset an actively running host controller results in undefined behavior.</p> <p>Device Mode:</p> <p>When software writes a 1 to this bit, the OTG controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. Writing a 1 to this bit in device mode is not recommended.</p>
RS	<p>Run/Stop.</p> <p>Host Mode:</p> <p>When set to a 1, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set to 1. When this bit is set to 0, the host controller completes the current transaction on the USB and then halts. The HC halted bit in the status register indicates when the host controller has finished the transaction and has entered the stopped state. Software should not write a 1 to this field unless the controller is in the halted state (i.e. HCHalted in the USBSTS register is a 1).</p> <p>Device Mode:</p> <p>Writing a 1 to this bit causes the OTG controller to enable a pull-up on D+ and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up becomes disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the OTG controller has been properly initialized. Writing a 0 to this causes a detach event.</p> <p>1 Run. 0 Stop.</p>

39.2.1.4.2 USB Status Register (USBSTS)

This register indicates various states of each module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them (indicated by a W1C in the bit's W cell in the figure).

Offset: Base + 144h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TI1	TI0	0	0	0	0	UPI	UAI	0	NAKI
W							W1C	W1C					W1C	W1C		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH	0	ULPII	0	SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
W						W1C		W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0*	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-18. USB Status Register (USBSTS)

Table 39-18. USBSTS Register Field Descriptions

Field	Description
TI1	General Purpose Timer Interrupt 1 (GPTINT1). This bit is set when the counter in the GPTIMER1CTRL (Non-EHCI) register transitions to 0. Writing a 1 to this bit clears it.
TI0	General Purpose Timer Interrupt 0 (GPTINT0). This bit is set when the counter in the GPTIMER0CTRL (Non-EHCI) register transitions to 0. Writing a 1 to this bit clears it.
UPI	USB Host Periodic Interrupt (USBHSTPERINT). This bit is set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the periodic schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the periodic schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. This bit is not used by the device controller and is always zero.
UAI	USB Host Asynchronous Interrupt (USBHSTASYNCINT). This bit is set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the asynchronous schedule. This bit is also set by the Host when a short packet is detected and the packet is on the asynchronous schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. The device controller does not use this bit and it is always zero.

Table 39-18. USBSTS Register Field Descriptions (continued)

Field	Description
NAKI	NAK Interrupt Bit. This bit is read-only. It is set by hardware when for a particular endpoint both the TX/RX endpoint NAK bit and the corresponding TX/RX endpoint NAK enable bit are set. This bit is automatically cleared by hardware when the all the enabled TX/RX endpoint NAK bits are cleared.
AS	Asynchronous Schedule Status. This bit reports the current real status of the asynchronous schedule. The controller is not required to immediately disable or enable the asynchronous schedule when software transitions the asynchronous schedule enable bit in the USBCMD register. When this bit and the asynchronous schedule enable bit are the same value, the asynchronous schedule is enabled (1) or disabled (0). Used only in host mode. 1 Enabled. 0 Disabled.
PS	Periodic Schedule Status. This bit reports the current real status of the periodic schedule. The controller is not required to immediately disable or enable the periodic schedule when software transitions the periodic schedule enable bit in the USBCMD register. When this bit and the periodic schedule enable bit are the same value, the periodic schedule is enabled (1) or disabled (0). Used only in host mode. 1 Enabled. 0 Disabled.
RCL	Reclamation. This is a status bit that detects an empty asynchronous schedule. Used only by the host mode. 1 Empty asynchronous schedule. 0 Non-empty asynchronous schedule.
HCH	HCHalted. This is a zero bit when the run/stop bit is a 1. The controller sets this bit to 1 after it has stopped executing because of the run/stop bit being set to 0, by software or the host controller hardware (e.g. internal error). Used only in host mode. 1 Halted. 0 Running.
ULPII	ULPI Interrupt. When the ULPI Viewport is present in the design, an event completion sets this interrupt.
SLI	DCSuspend. This is a non-EHCI bit present on the OTG module only. When a device controller enters a suspend state from an active state, this bit is set. This bit is only cleared by software writing a 1 to it.. Only used by the device controller. 1 Suspended. 0 Active.
SRI	Host mode:* This is a non-EHCI status bit. In host mode, this bit is set every 125us, provided the PHY clock is present and running (e.g. the port is NOT suspended). The host controller driver can use it as a time base. Device mode: SOF Received. When the OTG controller detects a Start Of (micro) Frame, this bit is be set to a 1. When a SOF is extremely late, the OTG controller automatically sets this bit to indicate an SOF was expected. Therefore, this bit is set roughly every 1 msec in device FS mode and every 125 msec in HS mode and is synchronized to the actual SOF received. Since the OTG controller is initialized to FS before connect, this bit is set at an interval of 1 msec during the prelude to the connect and chirp. Software writes a 1 to this bit to clear it.
URI	USB Reset Received. This is a non-EHCI bit present on the OTG module only. When the OTG controller detects a USB Reset and enters the default state, this bit is set to a 1. Software can write a 1 to this bit to clear the USB Reset Received status bit. Only used by the device mode. 1 Reset received. 0 No reset received.
AAI	Interrupt on Async Advance. System software can force the controller to issue an interrupt the next time the controller advances the asynchronous schedule by writing a 1 to the interrupt on async advance doorbell bit in the USBCMD register. This status bit indicates the assertion of that interrupt source. Only used by the host mode. 1 Async advance interrupt. 0 No async advance interrupt.

Table 39-18. USBSTS Register Field Descriptions (continued)

Field	Description
SEI	System Error. This bit is set when an error is detected on the system bus. If the system error enable (SEE) bit in the USBINTR is set, an interrupt is generated. The interrupt and status bits remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, the RUN/STOP (RS) bit of the USBCMD register is cleared, effectively disabling the controller. For the OTG controller in device mode, an interrupt is generated, but no other action is taken. 1 Error. 0 Normal operation.
FRI	Frame List Rollover. The controller sets this bit to 1 when the frame list index rolls over from its maximum value to 0. The exact value at which the rollover occurs depends on the frame list size. For example, if the frame list size (as programmed in the frame list size field of the USBCMD register) is 1024, the frame index register rolls over every time FRINDEX [1:3] toggles. Similarly, if the size is 512, the controller sets this bit to a 1 every time FHINDEX [12] toggles. Used only by the host mode.
PCI	Host mode: Port Change Detect. The controller sets this bit to 1 when on any port a connect status occurs, a port enable/disable change occurs, or the force port resume bit is set as the result of a J-K transition on the suspended port. Device mode: The OTG controller sets this bit to 1 when it enters the full or high-speed operational state. When the USB exits full or high-speed operation states due to reset or suspend events, the notification mechanisms are the USB reset received bit and the DCSuspend bits respectively. The device controller detects resume signalling only. This bit is not EHCI compatible.
UEI (USBERRINT)	USB Error Interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the controller. This bit is set along with the USBINT bit, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Table 39-99 in this chapter for more information on device error matrix. All others are ignored. 1 Error detected. 0 No error.
UI (USBINT)	USB Interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the Transfer Descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

39.2.1.4.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB Status register (USBSTS) continues to show interrupt sources even if the USBINTR register disables them, allowing polling of interrupt events by the software.

Offset: Base + 148h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TIE1	TIE0	0	0	0	0	UPIA	UAIE	0	NAKE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	ULPIE	0	SLE	SRE	URE	AAE	SEE	FRE	PCIE	UEE	UE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-19. USB Interrupt Enable Register (USBINTR)

Table 39-19. USBINTR Register Field Descriptions

Field	Description
TIE1	General Purpose Timer Interrupt Enable 1 When this bit is a 1 and the GPTINT1 bit in the USBSTS register is also a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the GPTINT1 bit.
TIE0	General Purpose Timer Interrupt Enable 0 When this bit is a 1 and the GPTINT0 bit in the USBSTS register is also a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the GPTINT0 bit.
UPIA	USB Host Periodic Interrupt Enable When this bit is a 1 and the USBHSTPERINT bit in the USBSTS register is also a 1, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBHSTPERINT bit.
UAIE	USB Host Async. Interrupt Enable When this bit is a 1 and the USBHSTASYNCINT bit in the USBSTS register is also a 1, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBHSTASYNCINT bit.
NAKE	NAK Interrupt Enable Software sets this bit if it wants to enable the hardware interrupt for the NAK Interrupt bit. If this bit and the corresponding NAK Interrupt bit are set, a hardware interrupt is generated.
ULPIE	ULPI Enable When this bit is a 1 and the ULPI interrupt bit in the USBSTS register transitions, the controller issues an interrupt. The interrupt is acknowledged by software writing a 1 to the ULPI Interrupt bit. Used by both host and device controller.

Table 39-19. USBINTR Register Field Descriptions (continued)

Field	Description
SLE	Sleep Enable. This is a non-EHCI bit. When this bit is a 1 and the DCSuspend bit in the USBSTS register transitions, the OTG controller issues an interrupt. The interrupt is acknowledged by software writing a 1 to the DCSuspend bit. Used only in device mode. 1 Enable. 0 Disable.
SRE	SOF Received Enable. This is a non-EHCI bit. When this bit is a 1 and the SOF Received bit in the USBSTS register is a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the SOF received bit. 1 Enable. 0 Disable.
URE	USB Reset Enable. This is a non-EHCI bit present on the OTG module only. When this bit is a 1 and the USB reset received bit in the USBSTS register is a 1, the device controller issues an interrupt. The interrupt is acknowledged by software clearing the USB reset received bit. Used only in device mode. 1 Enable. 0 Disable.
AAE	Interrupt on Async Advance Enable. When this bit is a 1 and the interrupt on async advance bit in the USBSTS register is a 1, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the interrupt on async advance bit. Used only in host mode. 1 Enable. 0 Disable.
SEE	System Error Enable. When this bit is a 1 and the system error bit in the USBSTS register is a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the system error bit. 1 Enable. 0 Disable.
FRE	Frame List Rollover Enable. When this bit is a 1 and the frame list rollover bit in the USBSTS register is a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the frame list rollover bit. Used only by the host mode. 1 Enable. 0 Disable.
PCE	Port Change Detect Enable. When this bit is a 1 and the port change detect bit in the USBSTS register is a 1, the controller issues an interrupt. The interrupt is acknowledged by software clearing the port change detect bit. 1 Enable. 0 Disable.
UEE	USB Error Interrupt Enable. When this bit is a 1 and the USBERRINT bit in the USBSTS register is a 1, the controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBERRINT bit in the USBSTS register. 1 Enable. 0 Disable.
UE	USB Interrupt Enable. When this bit is a 1 and the USBINT bit in the USBSTS register is a 1, the OTG controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing the USBINT bit. 1 Enable. 0 Disable.

39.2.1.4.4 Frame Index Register (FRINDEX)

In host mode, the controller uses this register to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N–3] select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the frame list size field in the USBCMD register.

This register must be written as a 32-bit word. Byte writes produce undefined results. This register cannot be written unless the OTG controller is in the halted state as indicated by the HCHalted bit. A write to this register while the run/stop bit is set to a 1 produces undefined results. Writes to this register also affect the SOF value.

In device mode, this register is read-only, and the OTG controller updates the FRINDEX[13–3] register from the frame number indicated by the SOF marker. When a SOF is received by the USB bus, FRINDEX[13–3] is checked against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is set to 0 (that is, SOF for 1 msec frame). If FRINDEX[13–3] is equal to the SOF value, FRINDEX[2–0] is incremented (that is, SOF for 125 μ sec microframe.)

Table 39-20 illustrates values of N based on the value of the frame list size in the USBCMD register when used in host mode.

Table 39-20. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N Value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

Universal Serial Bus Interface with On-The-Go

Offset: Base + 14Ch

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	FRINDEX													
W																
Reset	0	0	n	n	n	n	n	n	n	n	n	n	n	n	n	n
			= Unimplemented or Reserved													

Figure 39-20. USB Frame Index Register (FRINDEX)

Table 39-21. FRINDEX Register Field Descriptions

Field	Description
FRINDEX	Frame index. The value in this register increments at the end of each time frame (for example, microframe). Bits [N– 3] are used for the frame list current index. This means that each location of the frame list is accessed 8 times (frames or microframes) before moving to the next index. The value is the current frame number of the last frame transmitted. It is not used as an index. Bits 2–0 indicate the current microframe.

39.2.1.4.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on the MPC5121e.

39.2.1.4.6 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the Periodic Frame List in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the frame index register (FRINDEX) to enable the controller to step through the periodic frame list in sequence.

On the OTG module, this register is shared between the host and device mode functions. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See [Section 39.2.1.4.7, “Device Address Register \(DEVICEADDR\)—Non-EHCI,”](#) for more information.

Offset: Base + 154h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PERBASE															
W																
Reset	0	0	0	0	0	0	0	On	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PERBASE (con't)				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-21. Periodic Frame List Base Address Register (PERIODICLISTBASE)

Table 39-22. PERIODICLISTBASE Register Field Descriptions

Field	Description
PERBASE	Base Address. These bits correspond to memory address signal [31:12]. Only used in the host mode.

39.2.1.4.7 Device Address Register (DEVICEADDR)—Non-EHCI

This register is not defined in the EHCI specification. For the OTG module in device mode, the upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address matches all incoming addresses. Software reprograms the address after receiving a SET_ADDRESS descriptor.

The USBADRA can accelerate the SET_ADDRESS sequence by allowing the DCD to preset the USBADR register before the status phase of the SET_ADDRESS descriptor.

On the OTG module, this register is shared between the host and device mode functions. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See [Section 39.2.1.4.6, “Periodic Frame List Base Address Register \(PERIODICLISTBASE\),”](#) for more information.

Offset: Base + 154h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	USBADR							USBADRA	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-22. Device Address Register (DEVICEADDR)

Table 39-23. DEVICEADDR Register Field Descriptions

Field	Description
USBADR	Device Address. This field corresponds to the USB device address.
USBADRA	<p>Device Address Advance. Default=0. When this bit is 0, any writes to USBADR are instantaneous. When this bit is written to a 1 at the same time or before USBADR is written, the write to the USBADR field is staged and held in a hidden register. After an IN occurs on endpoint 0 and is ACKed, USBADR is loaded from the holding register.</p> <p>Hardware automatically clears this bit on the following conditions:</p> <ol style="list-style-type: none"> 1)IN is ACKed to endpoint 0. (USBADR is updated from staging register). 2)OUT/SETUP occur to endpoint 0. (USBADR is not updated). 3)Device Reset occurs (USBADR is reset to 0). <p>Note: After the status phase of the SET_ADDRESS descriptor, the DCD has 2 ms to program the USBADR field. This mechanism ensures this specification is met when the DCD cannot write to the device address within 2ms from the SET_ADDRESS status phase. If the DCD writes the USBADR with USBADRA equaling 1 after the SET_ADDRESS data phase (before the prime of the status phase), the USBADR is programmed instantly at the correct time and meets the 2ms USB requirement.</p>

39.2.1.4.8 Current Asynchronous List Address Register (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits [4–0] of this register cannot be modified by the system software and always return zeros when read.

On the OTG module, this register is shared between the host and device mode functions. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the ENDPOINTLISTADDR register. See [Section 39.2.1.4.9, “Endpoint List Address Register \(ENDPOINTLISTADDR\)— Non-EHCI,”](#) for more information.

Offset: Base + 158h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ASYBASE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ASYBASE (con't)												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-23. Current Asynchronous List Address Register (ASYNCLISTADDR)

Table 39-24. ASYNCLISTADDR Register Field Descriptions

Field	Description
ASYBASE	Link Pointer Low (LPL). These bits correspond to memory address signal [31:5]. This field may only reference a queue head (QH). Only used by the host controller.

39.2.1.4.9 Endpoint List Address Register (ENDPOINTLISTADDR)— Non-EHCI

This register is not defined in the EHCI specification. For the OTG module in device mode, this register contains the address of the top of the endpoint list in system memory. Bits [10–0] of this register cannot be modified by the system software and always return zeros when read. The memory structure referenced by this physical memory pointer is assumed to be 64 bytes. The queue head is actually a 48-byte structure, but must be aligned on a 64-byte boundary. However, the ENDPOINTLISTADDR[EPBASE] has a granularity of 2 Kbytes, so in practice the queue head should be 2-Kbyte aligned.

On the OTG module, this register is shared between the host and device mode functions. In device mode, it is the ENDPOINTLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See [Section 39.2.1.4.8, “Current Asynchronous List Address Register \(ASYNCLISTADDR\),”](#) for more information.

Offset: Base + 158h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EPBASE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EPBASE(con't)				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-24. Endpoint List Address Register (ENDPOINTLISTADDR)

Table 39-25. ENDPOINTLISTADDR Register Field Descriptions

Field	Description
EPBASE	Endpoint List Address. Address of the top of the endpoint list.

39.2.1.4.10 Host Controller Embedded TT Asynchronous Buffer Status

This register contains parameters for internal TT operations. This register is not used in device controller operation

Offset: Base + 15Ch

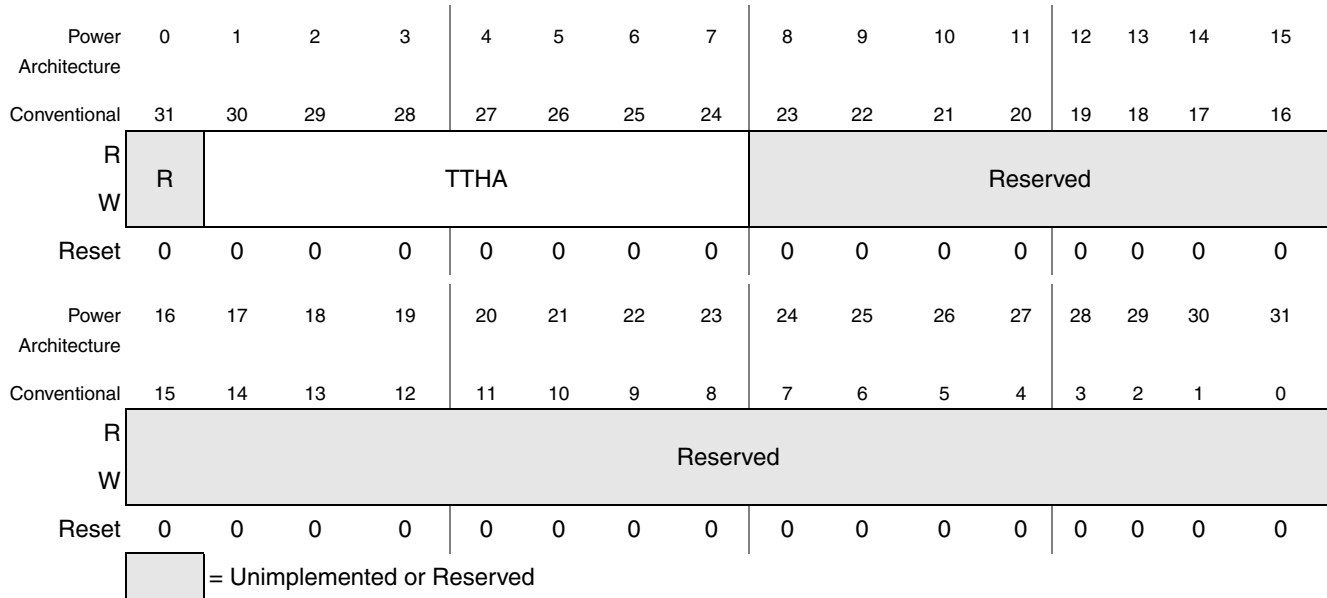


Figure 39-25. Host Controller Embedded TT Asynchronous Buffer Status

Table 39-26. TTCTRL Register Field Descriptions

Field	Description
TTHA	Internal TT Hub Address Representation. Default is 0 (read/write). This field matches against the hub address field in QH and siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the hub address in the QH or siTD does not match this address, the packet is broadcast on the high speed ports destined for a downstream high speed hub with the address in the QH/siTD.

39.2.1.4.11 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls and dynamically changes the burst size used during data movement on the initiator (master) interface.

Offset: Base + 160h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXPBURST								RXPBURST							
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

= Unimplemented or Reserved

Figure 39-26. Master Interface Data Burst Size Register (BURSTSIZE)

Table 39-27. BURSTSIZE Register Field Descriptions

Field	Description
TXPBURST	Programmable TX Burst Length. This register represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. If field AHBBRST of register SBUSCFG(090h) is different from zero, this field TXPBRUST returns the value of the INCRx length.
RXPBURST	Programmable RX Burst Length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. If field AHBBRST of register SBUSCFG(090h) is different from zero, this field RXPBRUST returns the value of the INCRx length.

39.2.1.4.12 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

This register is not defined in the EHCI specification. This register controls and dynamically changes the burst size used during data movement on device DMA transfers. It is only used in host mode.

The fields in this register control performance tuning associated with how the module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

T_0 = Standard packet overhead

T_I = Time to send data payload

T_s = Total Packet Flight Time (send-only) packet ($T_s = T_0 + T_I$)

T_{ff} = Time to fetch packet into TX FIFO up to specified level.

T_p = Total Packet Time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure T_p remains before end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at anytime during the pre-fill operation the time remaining in the [micro]frame is $< T_s$, packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to show the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that begins after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and should be minimized (not necessarily eliminated). Use of the TSCHEALTH (T_{ff}) parameter described below minimizes back-offs.

Offset: Base + 164h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	TXFIFOTHRES					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	TXSCHHEALTH					0	TXSCHOH						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-27. Transmit FIFO Timing Controls (TXFILLTUNING) Register

Table 39-28. TXFILLTUNING Register Field Descriptions

Field	Description
TXFIFOTHRES	FIFO Burst Threshold. This register controls the number of data bursts posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is two and this value should be as low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if the stream disable bit in USBMODE register is set.
TXSCHHEALTH	Scheduler Health Counter. These bits increment when the OTG controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter, and this counter maxes at 31.
TXSCHOH	Scheduler Overhead. These bits add an additional fixed offset to the schedule time estimator described above as T_{ff} . As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267 μ s when a device is connected in high-speed mode. The time unit represented in this register is 6.333 μ s when a device is connected in low/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $TXFIFOTHRES * (BURSTSIZE * 4 \text{ bytes-per-word}) / (40 * TimeUnit)$, always rounded to the next higher integer. <i>TimeUnit</i> is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, set TXSCHOH to $5 * (8 * 4) / (40 * 1.267) = 4$ for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

39.2.1.4.13 ULPI Viewport Register (ULPIVIEWPORT)

This register provides indirect access to the ULPI PHY register set. Although the core performs access to the ULPI PHY register set, extraordinary circumstances may exist where software may need direct access.

CAUTION

Writes to the ULPI through the viewport can substantially harm standard USB operations. Read operation should have no harmful side-effects to standard USB operations.

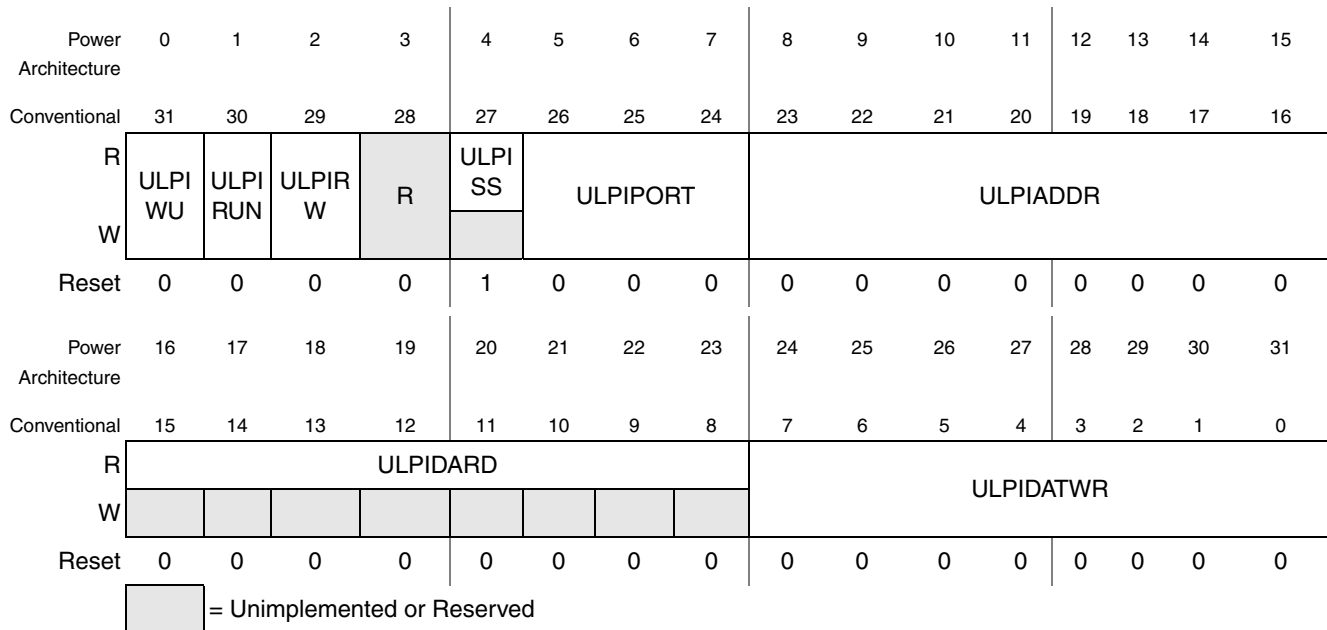
Two operations, wake-up and read/write, can be performed with the ULPI Viewport. Wake-up operation puts the ULPI interface into normal operation mode and re-enables the clock if necessary. A wakeup operation is required before accessing the registers when ULPI interface is operating in low-power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync. state bit (ULPISS). If this bit is a 1, ULPI interface is running in normal operation mode and can accept read/write operations. If the ULPISS indicates a 0, read/write operations are not able to execute. Undefined behavior results if ULPISS equals 0 and a read or write operation is performed. To execute a wakeup operation, write all 32-bits of the ULPI Viewport where ULPIPORT is constructed appropriately and the ULPIWU bit is 1 and ULPIRUN bit is 0. Poll the ULPI viewport until ULPIWU is zero for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI Viewport where ULPIDATWR, ULPIADDR, ULPIPORT, and ULPIRW are constructed appropriately and the ULPIRUN bit is 1. Poll the ULPI Viewport until ULPIRUN is zero for the operation to complete. After ULPIRUN is zero, ULPIDATRD is valid if the operation was a read.

The polling method above could also be replaced by an interrupt driven routine using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wakeup or read/write operation complete, the ULPI interrupt is set.

There are several optional features that system software may need to enable or disable as part of system configuration. These bits are contained in the interface and OTG control registers of the ULPI PHY register set. These registers also contain bits controlled by the link dynamically and therefore should only be modified by system software using the set/clear access method. Direct writes to these registers could have harmful side effects to the standard USB operations. The optional bits are as follows: bits 3 through 7 in the interface control register and bits 6 and 7 in the OTG control register. Refer to the ULPI Specification Revision 1.1 for further information on the use of the optional features.

Offset: Base + 170h


Figure 39-28. ULPI VIEWPORT Register (ULPIVIEWPORT)
Table 39-29. ULPIVIEWPORT Register Field Descriptions

Field	Description
ULPIWU	ULPI Wakeup. Writing the 1 to this bit begins the wakeup operation. The bit automatically transitions to 0 after the wakeup is complete. After this bit is set, the driver cannot set it back to 0. Note: The driver must never execute a wake-up and a read/write operation at the same time.
ULPIRUN	ULPI Read/Write Run. Writing the 1 to this bit begins the read/write operation. The bit automatically transitions to 0 after the read/write is complete. After this bit is set, the driver cannot set it back to 0. Note: The driver must never execute a wake-up and a read/write operation at the same time.
ULPIRW	ULPI Read/Write Control. This bit selects between running a read or a write operation 0> read 1> write
ULPISS	ULPI Sync State. This bit represents the state of the ULPI interface. Before reading this bit, the ULPIPORT field should be set accordingly if used with the multi-port host. Otherwise, this field should always remain 0. 1> Normal sync state 0> In another state (e.g. carkit, low power)
ULPIPORT	ULPI Port Number. For the wakeup or read/write operation to be executed, this value selects the port number the ULPI PHY is attached to in the multi-port host. The range is 0 to 7. This field should always be written as a 0 for the non-multi port products.
ULPIADDR	ULPI Data Address—When a read or write operation is commanded, the address of the operation is written to this field.
ULPIDATRD	ULPI Data Read Value—After a read operation completes, the result is placed in this field.
ULPIDATWR	ULPI Data Write Value—When a write operation is commanded, the data to be sent is written to this field.

39.2.1.4.14 Endpoint NAK (ENDPTNAK)

Offset: Base + 178h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													EPTN			
W													W1C			
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													EPRN			
W													W1C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1


 = Unimplemented or Reserved

Figure 39-29. Endpoint NAK (ENDPTNAK)

Table 39-30. Endpoint NAK Register Field Descriptions

Field	Description
EPTN	TX Endpoint NAK—Each TX endpoint has 1 bit in this field. The bit is set when the device sends a NAK handshake on a received IN token for the corresponding endpoint. EPTN[3] - Endpoint #3 EPTN[2] - Endpoint #2 EPTN[1] - Endpoint #1 EPTN[0] - Endpoint #0
EPRN	RX Endpoint NAK—Each RX endpoint has 1 bit in this field. The bit is set when the device sends a NAK handshake on a received OUT or PING token for the corresponding endpoint. EPRN[3] - Endpoint #3 EPRN[2] - Endpoint #2 EPRN[1] - Endpoint #1 EPRN[0] - Endpoint #0

39.2.1.4.15 Endpoint NAK Enable

Offset: Base + 17Ch

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													EPTNE			
W													W1C			
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													EPRNE			
W													W1C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 39-30. Endpoint NAK Enable Register (ENDPTNAKEN)

Table 39-31. Endpoint NAK Enable Register Field Descriptions

Field	Description
EPTNE	TX Endpoint NAK—Each bit is an enable bit for the corresponding TX Endpoint NAK bit. If this bit is set and the corresponding TX Endpoint NAK bit is set, the NAK Interrupt bit is set. EPTNE[3] - Endpoint #3 EPTNE[2] - Endpoint #2 EPTNE[1] - Endpoint #1 EPTNE[0] - Endpoint #0
EPRNE	RX Endpoint NAK—Each bit is an enable bit for the corresponding RX Endpoint NAK bit. If this bit is set and the corresponding RX Endpoint NAK bit is set, the NAK Interrupt bit is set. EPRNE[3] - Endpoint #3 EPRNE[2] - Endpoint #2 EPRNE[1] - Endpoint #1 EPRNE[0] - Endpoint #0

39.2.1.4.16 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of 0x0000_0001 to indicate all port routings default to this host controller.

Offset: Base + 180h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

= Unimplemented or Reserved

Figure 39-31. Configure Flag Register (CONFIGFLAG)

Table 39-32. CONFIGFLAG Register Field Descriptions

Field	Description
—	Reserved. (0x0000_0001, all port routings default to this host)

39.2.1.4.17 Port Status and Control Registers (PORTSC_n)

The OTG module has one port status and control register. The number of port registers implemented by a particular instantiation of a host controller is documented in the HCSPARAMs register. Software uses this information as an input parameter to determine how many ports need service. This register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power bit to one.

For the OTG module in device mode, the OTG controller does not support power control. Port control in device mode is only for status port reset, suspend, and current connect status. It also initiates test mode or forces signaling and allows software to put the PHY into low-power suspend mode and disable the PHY clock.

Offset: Base + 184h + (4*(Port Number – 1))

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS		0	PTW	PSPD		0	PFSC	PHCD	WKOC	WKDS	WKC N	PTC			
W																
Reset	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		PO		PP		LS		HSP	PR		SUSP	FPR	OCC	OCA	PE C
W														W1C		W1 C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

= Unimplemented or Reserved

Figure 39-32. Port Status and Control Register (PORTSCn)

Table 39-33. PORTSCn Register Field Descriptions

Field	Description
PTS	Port Transceiver Select. This register bit controls which parallel transceiver interface is selected. 00 UTMI parallel interface 01 Reserved 10 ULPI parallel interface 11 FS/LS serial interface.
STS	Reserved. Was STS Serial Transceiver Select (STS). There are no Serial Interface Engines in this implementation.
PTW	Parallel Transceiver Width. This register bit controls the data bus width of the parallel transceiver interface. This bit defaults to 1 after reset. 0 8-bit interface—[60 MHz] UTMI interface. 1 16-bit interface—[30 MHz] UTMI interface. PTW is only valid for UTMI mode (PTS = 00). This bit is not defined in the EHCI specification.
PSPD	Port Speed. This read-only register field indicates the speed at which the port is operating. This bit is not defined in the EHCI specification. 00 Full-speed 01 Low-speed 10 High-speed 11 Undefined
PFSC	Port Force Full-speed Connect. This bit disables the chirp sequence that allows the port to identify itself as an HS port. This is useful for testing FS configurations with an HS host, hub, or device. 0 Allow the port to identify itself as High Speed. 1 Force the port to only connect at Full-speed. This bit is not defined in the EHCI specification. This bit is for debugging purposes.

Table 39-33. PORTSC_n Register Field Descriptions (continued)

Field	Description
PHCD	<p>PHY Low Power Suspend. This bit is not defined in the EHCI specification.</p> <p>In host mode, the PHY can be put into low power suspend when the downstream device has been put into suspend mode or when no downstream device is connected. Low-power suspend is completely under the control of software.</p> <p>For the OTG module in device mode, the PHY can be put into low power suspend when the device is not running (USBCMD Run/Stop=0b) or suspend signaling is detected on the USB. Low-power suspend is cleared automatically when the resume signaling has been detected or when forcing port resume.</p> <p>0 Normal PHY operation. 1 Signal the PHY to enter low power suspend mode</p> <p>Reading this bit indicates the status of the PHY.</p>
WKOC	<p>Wake On Over-Current Enable. Writing this bit to 1 enables the port to be sensitive to over-current conditions as wake-up events.</p> <p>This field is zero if Port Power (PP) is zero.</p> <p>This bit is (OTG/host mode only) for use by an external power control circuit.</p>
WKDS	<p>Wake On Disconnect Enable. Writing this bit to 1 enables the port to be sensitive to device disconnects as wake-up events.</p> <p>This field is zero if Port Power(PP) is zero or in device mode.</p> <p>This bit is (OTG/host mode only) for use by an external power control circuit.</p>
WKN	<p>Wake On Connect Enable. Writing this bit to 1 enables the port to be sensitive to device connects as wake-up events.</p> <p>This field is zero if Port Power(PP) is zero or in device mode.</p>
PTC	<p>Port Test Control. Any value other than zero indicates the port is operating in test mode.</p> <p>0000 Not Enabled. 0001 J_STATE. 0010 K_STATE. 0011 SE0_NAK. 0100 Packet. 0101 FORCE_ENABLE. 0110-1111 Reserved.</p> <p>Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.</p>
PIC	<p>Port Indicator Control. These bits control the link indicator signals. These signals are valid for host mode only.</p> <p>00 Off. 01 Amber. 10 Green. 11 Undefined.</p> <p>Refer to the USB Specification Revision 2.0 [3] for a description on how these bits are to be used.</p> <p>This field is output from the module on the USB port control signals for use by an external LED driving circuit.</p>
PO	<p>Port Owner. This bit unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 when the configured bit is zero. System software uses this field to release ownership of the port to a selected module (in the event that the attached device is not a high-speed device). Software writes a 1 to this bit when the attached device is not a high-speed device. A one in this bit means an internal companion controller owns and controls the port.</p> <p>Port owner hand-off is not implemented in this design, therefore this bit is always 0.</p>
PP	<p>Port Power. This bit represents the current setting of the switch (0=off, 1=on). When power is not available on a port(i.e. PP equals a 0), the port is non-functional and does not report attaches, detaches, etc.</p> <p>When an over-current condition is detected on a powered port, the PP bit in each affected port is transitioned by the host controller driver from a one to a zero (removing power from the port).</p> <p>This feature is implemented in the host/OTG controller (PPC = 1).</p> <p>For the OTG module in a device-only implementation, port power control is not necessary. Therefore, PPC and PP equal 0.</p>

Table 39-33. PORTSCn Register Field Descriptions (continued)

Field	Description
LS	Line Status. These bits reflect the current logical levels of the USB D+ (bit 11) and D– (bit 10) signal lines. The use of line status by the host controller driver is not necessary (unlike EHCI) because hardware manages the connection of FS and LS. 00 SE0. 01 J-state. 10 K-state. 11 Undefined.
HSP	High Speed Port. This bit is redundant with the PORTSCn.PSPD bits. 1 Host/Device connected is operating in High Speed mode. 0 Host/Device connected is not operating in High Speed mode. This field is zero if Port Power(PP) is zero.
PR	Port Reset. In host mode, when software writes a one to this bit the bus-reset sequence as defined in the USB specification revision 2.0 is started. This bit automatically changes to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver. For the DR module in device mode, this bit is a read-only status bit. Device reset from the USB bus is also indicated in the USBSTS register. 1 Port is in Reset. 0 Port is not in Reset. This field is zero if Port Power(PP) is zero.
SUSP	Suspend In host mode: The port enabled bit (PE) and suspend (SUSP) bit define the port states as follows: 0x Disable. 10 Enable. 11 Suspend. When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Bit status does not change until the port is suspended and there may be a delay in suspending a port if there is a transaction currently in progress on the USB. The module unconditionally sets this bit to zero when software sets the force port resume bit to zero. The host controller ignores a write of zero to this bit. If host software sets this bit to a one when the port is not enabled (i.e. port enabled bit is a zero), results are undefined. This field is zero if port power(PP) is zero in host mode. For the OTG module in device mode: 1 Port in suspend state. 0 Port not in suspend state. Default. In device mode, this bit is a read only status bit.

Table 39-33. PORTSC_n Register Field Descriptions (continued)

Field	Description
FPR	<p>Force Port Resume. This bit is not-EHCI compatible.</p> <p>1 Resume detected/driven on port. 0 No resume (K-state) detected/driven on port.</p> <p>In host mode: Software sets this bit to one to drive resume signaling. The controller sets this bit to one if a J-to-K transition is detected while the port is in the suspend state. When this bit transitions to a one, a J-to-K transition is detected, and the port change detect bit in the USBSTS register is also set to one. This bit automatically changes to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver. When the controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (full-speed K) is driven on the port as long as this bit remains a one. This bit remains a one until the port has switched to the high-speed idle. Writing a zero has no effect because the port controller times the resume operation to clear the bit the port control state switches to HS or FS idle. This field is zero if port power(PP) is zero in host mode.</p> <p>In Device mode: After the device has been in suspend state for 5 msec or more, software must set this bit to one to drive resume signaling before clearing. the OTG controller sets this bit to one if a J-to-K transition is detected while the port is in the suspend state. The bit is cleared when the device returns to normal operation. Also, when this bit transitions to a one because a J-to-K transition is detected, the port change detect bit in the USBSTS register is also set to one.</p>
OCC	<p>Over-current Change. This bit gets set to one when there is a change to over-current active. Software clears this bit by writing a one to this bit position.</p> <p>For host/OTG implementations, you can provide over-current detection to the USB_n_PWRFAULT signal for this condition.</p> <p>For device-only implementations, this bit must always be 0.</p> <p>1 Over current detect. 0 No over current.</p>
OCA	<p>Over-current Active. This bit automatically transitions from one to zero when the over current condition is removed.</p> <p>For host/OTG implementations the user can provide over-current detection to the USB_n_PWRFAULT signal for this condition.</p> <p>For device-only implementations this bit must always be 0.</p> <p>1 Port currently in over-current condition. 0 Port not in over-current condition.</p>
PEC	<p>Port Enable/Disable Change.</p> <p>For the root hub, this bit is set to a one only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a one to it.</p> <p>In Device mode, the device port is always enabled. (This bit is zero).</p> <p>1 Port disabled. 0 No change.</p> <p>This field is zero if port power(PP) is zero.</p>
PE	<p>Port Enabled/Disabled.</p> <p>In host mode, ports can only be enabled by the controller as a part of the reset and enable. Software cannot enable a port by writing a one to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. The bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host and bus events.</p> <p>When the port is disabled, (0) downstream propagation of data is blocked except for reset.</p> <p>This field is zero if port power(PP) is zero in host mode.</p> <p>In device mode (DR-only), the device port is always enabled. (This bit is one).</p>

Table 39-33. PORTSC_n Register Field Descriptions (continued)

Field	Description
CSC	<p>Connect Change Status.</p> <p>In host mode, this bit indicates a change has occurred in the port's current connect status. The controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, and hub hardware is setting an already-set bit (the bit remains set). Software clears this bit by writing a one to it.</p> <p>1 Connect Status has changed. 0 No change.</p> <p>This field is zero if port power(PP) is zero. In device mode, this bit is undefined.</p>
CCS	<p>Current Connect Status.</p> <p>In host mode:</p> <p>1 Device is present 0 No device present.</p> <p>This field is zero if port power(PP) is zero in host mode.</p> <p>In device mode:</p> <p>1 Attached 0 Not attached.</p> <p>A one indicates the device successfully attached and is operating in either high-speed or full-speed as indicated by the high speed port bit in this register. A zero indicates the device did not attach successfully or was forcibly disconnected by the software writing a zero to the run bit in the USBCMD register. It does not state the device being disconnected or suspended.</p>

39.2.1.4.18 On-The-Go Status and Control (OTGSC)—Non-EHCI

This register is not defined in the EHCI specification. Both OTG modules implement one On-The-Go (OTG) Status and Control register. For more information please refer to the *On-The-Go Supplement to the USB 2.0 Specification*.

The OTGSC register has four sections:

- OTG Interrupt enables (Read/Write)
- OTG Interrupt status (Read/Write to Clear)
- OTG Status inputs (Read Only)
- OTG Controls (Read/Write)

The status inputs are de-bounced using a 1 msec time constant. Values on the status inputs that do not persist for more than 1 msec do not cause an update of the status inputs or cause an OTG interrupt.

Offset: Base + 1A4h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	DPIE	1msE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	0	DPIS	1msS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										W1C	W1C	W1C	W1C	W1C	W1C	W1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	DPS	1msT	BSE	BSV	ASV	AVV	ID	HABA	HADP	IDPU	DP	OT	HAA R	VC	VD
W																
Reset	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-33. On-The-Go Status and Control Register (OTGSC)

Table 39-34. OTGSC Register Field Descriptions (Sheet 1 of 3)

Field	Description
DPIE	Data Pulse Interrupt Enable. 1 Enable. 0 Disable.
1msE	1 millisecond timer Interrupt Enable. 1 Enable. 0 Disable.
BSEIE	B Session End Interrupt Enable. 1 Enable. 0 Disable.
BSVIE	B Session Valid Interrupt Enable. 1 Enable. 0 Disable.
ASVIE	A Session Valid Interrupt Enable. 1 Enable. 0 Disable.
AVVIE	A VBus Valid Interrupt Enable. 1 Enable. 0 Disable.
IDIE	USB ID Interrupt Enable. 1 Enable. 0 Disable.
DPIS	Data Pulse Interrupt Status. This bit is set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE.CM equals Host (11) and PORTSC(0).PortPower equals Off (0). Software must write a 1 to clear this bit.

Table 39-34. OTGSC Register Field Descriptions (Sheet 2 of 3)

Field	Description
1msS	1 millisecond timer Interrupt Status. This bit is set once every millisecond. Software must write a 1 to clear this bit.
BSEIS	B Session End Interrupt Status. This bit is set when VBus has fallen below the B session end threshold. Software must write a 1 to clear this bit.
BSVIS	B Session Valid Interrupt Status. This bit is set when VBus has risen above or fallen below the B session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
ASVIS	A Session Valid Interrupt Status. This bit is set when VBus has risen above or fallen below the A session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
AVVIS	A VBus Valid Interrupt Status. This bit is set when VBus has risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. Software must write a 1 to clear this bit.
IDIS	USB ID Interrupt Status. This bit is set when a change on the ID input is detected. Software must write a 1 to clear this bit.
DPS	Data Bus Pulsing Status. 1 Pulsing detected on port. 0 No pulsing on port.
1msT	1 millisecond timer toggle. This bit toggles once per millisecond.
BSE	B Session End. 1 VBus is below the B session end threshold. 0 VBus is above the B session end threshold.
BSV	B Session Valid. 1 VBus is above the B session valid threshold. 0 VBus is below the B session valid threshold.
ASV	A Session Valid. 1 VBus is above the A session valid threshold. 0 VBus is below the A session valid threshold.
AVV	A VBus Valid. 1 VBus is above the A VBus valid threshold. 0 VBus is below the A VBus valid threshold.
ID	USB ID. 1 B device. 0 A device.
HABA	Hardware Assist B-Disconnect to A-connect. 1 Enable automatic B-disconnect to A-connect sequence. 0 Disabled.
HADP	Hardware Assist Data-Pulse. 1 Start Data Pulse Sequence.
IDPU	ID Pullup. This bit provide control over the ID pull-up resistor: 0 off, 1 on When this bit is 0, the ID input is not sampled.

Table 39-34. OTGSC Register Field Descriptions (Sheet 3 of 3)

Field	Description
DP	Data Pulsing. 1 The pullup on DP is asserted for data pulsing during SRP. 0 The pullup on DP is not asserted.
OT	OTG Termination. This bit must be set when the OTG device is in device mode. 1 Enable pulldown on DM. 0 Disable pulldown on DM.
HAAR	Hardware Assist Auto-Reset. 0 Disabled. 1 Enable automatic reset after connect on host port.
VC	VBUS Charge. Setting this bit causes the VBus line to be charged. This is for VBus pulsing during SRP.
VD	VBUS discharge. Setting this bit causes VBus to discharge through a resistor.

39.2.1.4.19 USB Mode Register (USBMODE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls the operating mode of the module.

Offset: Base + 1A8h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0						
W											VBPS	SDIS	SLOM	ES	CM	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-34. USB Mode Register (USBMODE)**Table 39-35. USBMODE Register Field Descriptions (Sheet 1 of 3)**

Field	Description
VBPS	Vbus Power Select (0 - Output is 0; 1 - Output is one) This bit is connected to the vbus_pwr_select output and can be used for any generic control but is named to be used by logic that selects between an on-chip Vbus power source (charge pump) and an off-chip source in systems when both are available.

Offset: Base + 1A8h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0						
W											VBPS	SDIS	SLOM	ES	CM	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

Figure 39-34. USB Mode Register (USBMODE)

Table 39-35. USBMODE Register Field Descriptions (Sheet 2 of 3)

Field	Description
SDIS	<p>Stream Disable.</p> <p>In host mode, setting this bit to a 1 ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency is filled to capacity before the packet is launched onto the USB.</p> <p>Time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature.</p> <p>In systems with high system bus utilization, setting this bit ensures no overruns or underruns during operation at the expense of link utilization. For those who desire optimal link performance, SDIS can be left clear, and the rules used under the description of the TXFILLTUNING register to limit underruns/overruns.</p> <p>1 Active. 0 Inactive.</p> <p>In device mode, setting this bit to a 1 disables double priming on both RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet, the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems.</p> <p>In high speed mode, all packets received are responded to with a NYET handshake when stream disable is active.</p>
SLOM	<p>Setup Lockout Mode. For the OTG module in device mode, this bit controls behavior of the setup lock mechanism. See Section 39.8.5.2, "Control Endpoint Operation Model."</p> <p>1 Setup Lockouts Off(DCD requires use of Setup Data Buffer Tripwire in USBCMD). 0 Setup Lockouts On.</p>
ES	<p>Endian Select.</p> <p>This bit can change the byte ordering of the transfer buffers to match the host microprocessor bus architecture. The bit fields in the microprocessor interface and the DMA data structures (including the setup buffer within the device QH) are unaffected by the value of this bit, because they are based upon 32-bit words.</p> <p>0 Little Endian [Default]: first byte referenced in least significant byte of 32-bit word. 1 Big Endian - first byte referenced in most significant byte of 32-bit word.</p>

Offset: Base + 1A8h

Power	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Architecture																
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Architecture																
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0						
W											VBPS	SDIS	SLOM	ES	CM	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-34. USB Mode Register (USBMODE)

Table 39-35. USBMODE Register Field Descriptions (Sheet 3 of 3)

Field	Description
CM	<p>Controller Mode.</p> <p>This register can only be written once after reset. If necessary to switch modes, software must reset the controller by writing to the RESET bit in the USBCMD register before reprogramming this register.</p> <p>00 Idle (Default for combination host/device).</p> <p>01 Reserved.</p> <p>10 Device Controller (Default for device only controller).</p> <p>11 Host Controller (Default for host only controller).</p> <p>The OTG module defaults to the idle state and needs to be initialized to the desired operating mode after reset.</p>

39.2.1.4.20 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

This register is not defined in the EHCI specification. This register contains the endpoint setup status. It is used only by the OTG module in device mode.

Offset: Base + 1ACh

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	ENDPTSETUPSTAT			
W													W1C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-35. Endpoint Setup Status Register (ENDPTSETUPSTAT)

Table 39-36. ENDPTSETUPSTAT Register Field Descriptions

Field	Description
ENDPTSETUPSTAT	Endpoint Setup Status. For every setup transaction received, a corresponding bit in this register is set to 1. Software must clear or acknowledge the setup transfer by writing a 1 to a respective bit after it has read the setup data from queue head. The response to a setup packet as in the order of operations and total response time is crucial to limit bus timeouts while the setup lock our mechanism is engaged. See Managing Endpoints in the Device Operational Model. This register is used only in device mode.

39.2.1.4.21 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to initialize endpoints. It is used only by the OTG module in device mode.

Offset: Base + 1B0h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													PETB			
W													W1S			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													PERB			
W													W1S			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-36. Endpoint Initialization Register (ENDPTPRIME)

Table 39-37. ENDPTPRIME Register Field Descriptions

Field	Description
PETB	<p>Prime Endpoint Transmit Buffer. For each endpoint, a corresponding bit requests a buffer to prepare for a transmit operation to respond to a USB IN/INTERRUPT transaction. Software should write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware clears this bit when the associated endpoint(s) is (are) successfully primed.</p> <p>Note: Hardware momentarily sets these bits during hardware re-priming operations when a dTD is retired, and the dQH is updated.</p> <p>PETB[3] - Endpoint #3 PETB[2] - Endpoint #2 PETB[1] - Endpoint #1 PETB[0] - Endpoint #0</p>
PERB	<p>Prime Endpoint Receive Buffer. For each endpoint, a corresponding bit requests a buffer to prepare for a receive operation for when a USB host initiates a USB OUT transaction. Software should write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware clears this bit when the associated endpoint(s) is (are) successfully primed.</p> <p>Note: Hardware momentarily sets these bits during hardware re-priming operations when a dTD is retired, and the dQH is updated.</p> <p>PERB[3] - Endpoint #3 PERB[2] - Endpoint #2 PERB[1] - Endpoint #1 PERB[0] - Endpoint #0</p>

39.2.1.4.22 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI

This register is not defined in the EHCI specification. This register is used only by the OTG module in device mode.

Offset: Base + 1B0h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													FETB			
W													W1S			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													FERB			
W													W1S			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-37. Endpoint Flush Register (ENDPTFLUSH)

Table 39-38. ENDPTFLUSH Register Field Descriptions

Field	Description
FETB	Flush Endpoint Transmit Buffer. Writing a 1 to a bit(s) in this register causes the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful. FETB[3] - Endpoint #3 FETB[2] - Endpoint #2 FETB[1] - Endpoint #1 FETB[0] - Endpoint #0
FERB	Flush Endpoint Receive Buffer. Writing a 1 to a bit(s) causes the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful. FERB[3] - Endpoint #3 FERB[2] - Endpoint #2 FERB[1] - Endpoint #1 FERB[0] - Endpoint #0

39.2.1.4.23 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI

This register is not defined in the EHCI specification. This register is used only by the OTG module in device mode.

Offset: Base + 1B8h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													ETBR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													ERBR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-38. Endpoint Status Register (ENDPTSTATUS)

Table 39-39. ENDPTSTATUS Register Field Descriptions

Field	Description
ETBR	<p>Endpoint Transmit Buffer Ready. One bit for each endpoint indicates status of the respective endpoint buffer. Hardware sets this bit as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There is always a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register.</p> <p>Note: Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.</p> <p>ETBR[3] - Endpoint #3 ETBR[2] - Endpoint #2 ETBR[1] - Endpoint #1 ETBR[0] - Endpoint #0</p>
ERBR	<p>Endpoint Receive Buffer Ready. One bit for each endpoint indicates status of the respective endpoint buffer. Hardware sets this bit to a one by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There is always a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register.</p> <p>Note: Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.</p> <p>ERBR[3] - Endpoint #3 ERBR[2] - Endpoint #2 ERBR[1] - Endpoint #1 ERBR[0] - Endpoint #0</p>

39.2.1.4.24 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI

This register is not defined in the EHCI specification. This register is used only by the OTG module in device mode.

Offset: Base + 1BCh

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R													ETCE			
W													W1C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R													ERCE			
W													W1C			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-39. Endpoint Complete Register (ENDPTCOMPLETE)

Table 39-40. ENDPTCOMPLETE Register Field Descriptions

Field	Description
ETCE	Endpoint Transmit Complete Event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ETCE[3] - Endpoint #3 ETCE[2] - Endpoint #2 ETCE[1] - Endpoint #1 ETCE[0] - Endpoint #0
ERCE	Endpoint Receive Complete Event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ERCE[3] - Endpoint #3 ERCE[2] - Endpoint #2 ERCE[1] - Endpoint #1 ERCE[0] - Endpoint #0

39.2.1.4.25 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI

This register is not defined in the EHCI specification. Every device implements endpoint 0 as a control endpoint.

Offset: Base + 1C0h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	TXE	0	0	0	TXT		0	TXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RXE	0	0	0	RXT		0	RXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-40. Endpoint Control Register 0 (ENDPTCTRL0)

Table 39-41. ENDPTCTRL0 Register Field Descriptions (Sheet 1 of 2)

Field	Description
TXE	TX Endpoint Enable. Endpoint zero is always enabled. 1 Enable.
TXT	TX Endpoint type. Endpoint zero is always a control endpoint (00).
TXS	TX Endpoint Stall. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until the software clears this bit or it is automatically cleared upon receipt of a new SETUP request. After receiving a SETUP request, this bit continues to be cleared by hardware until the associated ENDPTSETUPSTAT bit is cleared. Note: There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems, it is unlikely the DCD software observes this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, follow this procedure: continually write this stall bit until it is set or until a new setup has been received by checking the associated endptsetupstat bit. 1 Endpoint Stalled. 0 Endpoint OK.
RXE	RX Endpoint Enable. Endpoint zero is always enabled. 1 Enabled.
RXT	RX Endpoint type. Endpoint zero is always a control endpoint (00).

Offset: Base + 1C0h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	TXE	0	0	0	TXT		0	TXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RXE	0	0	0	RXT		0	RXS
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 39-40. Endpoint Control Register 0 (ENDPTCTRL0)

Table 39-41. ENDPTCTRL0 Register Field Descriptions (Sheet 2 of 2)

Field	Description
RXS	<p>RX Endpoint Stall.</p> <p>Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears this bit or it is automatically cleared upon receipt of a new SETUP request.</p> <p>After receiving a SETUP request, hardware continues to clear this bit until the associated ENDPTSETUPSTAT bit is cleared.</p> <p>Note: There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems, it is unlikely the DCD software observes this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, follow this procedure: continually write this stall bit until it is set or until a new setup has been received by checking the associated ENDPTSETUPSTAT bit.</p> <p>1 Endpoint Stalled. 0 Endpoint OK.</p>

39.2.1.4.26 Endpoint Control Register n (ENDPTCTRL n)—Non-EHCI

These registers are not defined in the EHCI specification. There is an ENDPTCTRL n register of each endpoint in a device.

Offset: Base + 1C0h + (4*(Endpoint Number))

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	TXE	TXR	TXI	0	TXT	TXD	TXS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RXE	RXR	RXI	0	RXT	RXD	RXS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-41. Endpoint Control Register n (ENDPTCTRL n)

Table 39-42. ENDPTCTRL n Register Field Descriptions (Sheet 1 of 2)

Field	Description
TXE	TX Endpoint Enable. 1 Enabled. 0 Disabled.
TXR	TX Data Toggle Reset. When a configuration event is received for this endpoint, software must write a one to this bit to synchronize the data PIDs between the host and device.
TXI	TX Data Toggle Inhibit. This bit is used only for test and should always be written as zero. Writing a one to this bit causes this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 1 PID Sequencing Disabled. 0 PID Sequencing Enabled.
TXT	TX Endpoint type. 00 Control. 01 Isochronous. 10 Bulk. 11 Interrupt.
TXD	TX Endpoint Data Source. This bit should always be written as 0, which selects the Dual Port Memory/DMA Engine as the source.

Table 39-42. ENDPTCTRL n Register Field Descriptions (Sheet 2 of 2)

Field	Description
TXS	<p>TX Endpoint Stall. This bit is set automatically upon receipt of a setup request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt of a setup request if this endpoint is configured as a control endpoint.</p> <p>Software can write a one to this bit to force the endpoint to return a stall handshake to the host. It continues returning stall until this bit is cleared by software or automatically cleared as above.</p> <p>Software can write a one to this bit to force the endpoint to return a stall handshake to the Host. This control continues to stall until this bit is either cleared by software or automatically cleared as above for control endpoints.</p> <p>Note: [control endpoint types only] There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems, it is unlikely the DCD software observes this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, follow this procedure: continually write this stall bit until it is set or until a new setup has been received by checking the associated endptsetupstat bit.</p> <p>1 Endpoint Stalled. 0 Endpoint OK.</p>
RXE	<p>RX Endpoint Enable.</p> <p>1 Enabled. 0 Disabled.</p>
RXR	<p>RX Data Toggle Reset. When a configuration event is received for this endpoint, software must write a one to this bit to synchronize the data PIDs between the host and device.</p>
RXI	<p>RX Data Toggle Inhibit. This bit is used only for test and should always be written as zero. Writing a 1 to this bit causes this endpoint to ignore the data toggle sequence and always accepts data packets regardless of their data PID.</p> <p>1 PID Sequencing Enabled. 0 PID Sequencing Disabled.</p>
RXT	<p>RX Endpoint type.</p> <p>00 Control. 01 Isochronous. 10 Bulk. 11 Interrupt.</p>
RXD	<p>RX Endpoint Data Sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink.</p>
RXS	<p>RX Endpoint Stall. This bit is set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It is cleared automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint.</p> <p>Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until this bit is cleared by software or automatically cleared as above.</p> <p>Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. This control continues to STALL until this bit is cleared by software or automatically cleared as above for control endpoints.</p> <p>Note: [control endpoint types only] There is a slight delay (50 clocks max.) between the ENDPTSETUPSTAT being cleared and hardware continuing to clear this bit. In most systems, it is unlikely the DCD software observeS this delay. However, should the DCD observe that the stall bit is not set after writing a 1 to it, follow this procedure: continually write this stall bit until it is set or until a new setup has been received by checking the associated endptsetupstat bit.</p> <p>1 Endpoint Stalled. 0 Endpoint OK.</p>

39.2.1.4.27 USB General Control Register (USBGENCTRL)—Non-EHCI

This register is not defined in the EHCI specification. The general purpose control register is shown in Figure 39-42. This register uses big endian byte ordering.

Offset: Base + 200h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	WU_INT_CLR	ULPI_SEL	PPP	PFP	WU_ULPI_EN	WU_IE
W											W1C					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

□ = Unimplemented or Reserved

Figure 39-42. USB General Control Register (USBGENCTRL)

Table 39-43. USBGENCTRL Register Field Descriptions

Field	Description
WU_INT_CLR	Wake-up Interrupt clear. 1 Clear the wake-up interrupt. 0 Default, no action.
ULPI_SEL	USB0 ONLY. ULPI I/F Select (default the UTMI I/F is selected). This bit must be programmed to 1 when the ULPI I/F should be used. 1 Select the ULPI I/F. 0 Select the UTMI I/F.
PPP	USB0 ONLY. PORT POWER POLARITY. This bit allow to program the polarity of the DRVVBUS signal. 1 Invert the DRVVBUS signal. 0 The DRVVBUS is not inverted.
PFP	USB0 ONLY. POWER FAULT POLARITY. This bit allow to program the polarity of the PWR_FAULT signal. 1 Invert the PWR_FAULT signal. 0 The PWR_FAULT is not inverted.
WU_ULPI_EN	WAKEUP ON ULPI INTERRUPT EVENT. This bit is used to enable the wake up from the ULPI I/F. 1 Wake Up Interrupt Enabled. 0 Wake Up Interrupt Disabled.
WU_IE	WAKEUP INTERRUPT ENABLE. This bit is used to enable the low power wakeup interrupt. 1 Low power wakeup interrupt enabled. 0 Low power wakeup interrupt disabled.

39.2.1.4.28 On-Chip PHY Control Register (ISIPHYCTRL)—Non-EHCI

This register is not defined in the EHCI specification. The On-Chip PHY Control register is used to control the settings of the On-Chip UTMI+ PHY. The on-chip PHY is only available for USB0. It's contents is shown in Figure 39-43.

Offset: Base + 204h

Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0			BSENH	BSEN	LSFE	PXE
W											OCO	PHYE				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 39-43. On-Chip PHY Control Register (ISIPHYCTRL)

Table 39-44. ISIPHYCTRL Register Field Descriptions

Field	Description
OCO	Oscillator Control Override 1 ISIPHYCTRL.PXE (bit [0]) controls the enable/disable signal of the USB PHY Oscillator. 0 The USB PHY controls the enable/disable signal of the USB PHY Oscillator (suspendM).
PHYE	On-Chip UTMI PHY Enable 1 Enable On-Chip UTMI PHY 0 Disable On-Chip UTMI PHY
BSENH	Bit Stuff Enable High. 1 Enable bit stuffing on UTMI+ DATA bits [15:8]. 0 Disable bit stuffing.
BSEN	Bit Stuff Enable. 1 Enable bit stuffing on UTMI+ DATA bits [7:0]. 0 Disable bit stuffing.
LSFE	Line State Filter Enable. Enables filtering of LineState to account for skew between D-/D+ signals. 1 Line State Filter Enabled. 0 Line State Filter Disabled.
PXE	PHY Oscillator Enable. 1 PHY Oscillator Enabled. 0 PHY Oscillator Disabled.

39.3 Functional Description

Both modules, USB1/USB0, can be broken down into functional sub-blocks described below.

39.3.1 System Interface

The system interface block contains all the control and status registers that allow the CPU core to interface to the module. These registers allow the processor to control the configuration of the module, ascertain the capabilities of each module and control the module's operation.

39.3.2 DMA Engine

Both USB controllers contain local DMA engines. The DMA Engine interface is responsible for moving all of the data to transfer over the USB between the controller and buffers in system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol.

The DMA controllers must access both control information and packet data from system memory. The control information is contained in link-list based queue structures. The DMA controllers have state machines able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers performed by the host controller. In device mode, the data structures are similar to those in the EHCI specification and allow device responses to be queued for each of the active pipes in the device.

39.3.3 FIFO RAM Controller

The FIFO RAM controller is for context information and to control FIFOs between the protocol engine and the DMA controller. These FIFOs decouple the system processor/memory bus requests from the extremely tight timing required by USB.

The use of the FIFO buffers differs between host and device mode operation. In host mode, a single data channel is maintained in each direction through the buffer memory. In device mode, multiple FIFO channels are maintained for each of the active endpoints in the system.

For both modules (USB0 and USB1), device operation uses a single 256-byte RX buffer and a 512-byte TX buffer for each endpoint. The 512-byte buffers allow the modules to buffer a complete HS bulk packet.

The USB1 and USB0 module interfaces to ULPI compatible PHY. The USB0 module interfaces to an additional internal PHY. The primary function of the port controller block is to isolate the rest of the module from the transceiver and to move all of the transceiver signaling into the primary clock domain of the module. This allows the module to run synchronously with the system processor and its associated resources.

39.4 OTG Operations

39.4.1 Register Bits

In the previous section, the register interface has behaviors described for device and host mode. However, during OTG operations, it is necessary to perform tasks independent of the controller mode.

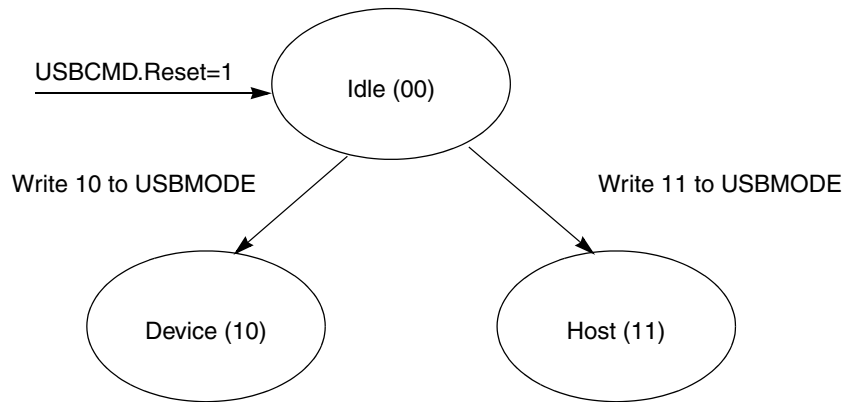


Figure 39-44. Controller Mode

Figure 39-44 also shows that the only way to transition the controller mode out of host or device mode is with the controller reset bit. Therefore, it is also necessary for the OTG tasks to be performed independent of a controller reset as well as independent of the controller mode.

To this end, the following list contains the register bits used for OTG operations, which are independent of the controller mode and are also not affected by a write to the reset bit in the USBCMD register.

- All Identification Registers
- All Device/Host Capability Registers
- OTGSC: All bits
- PORTSC:
 - Physical Interface Select
 - Physical Interface Serial Select
 - Physical Interface Data Width
 - Physical Interface Low Power
 - Physical Interface Wake Signals
 - Port Indicators
 - Port Power

39.4.2 Hardware Assist

The hardware assist provides automated response and sequencing that may not be possible to software with significant interrupt latency response times. The use of this additional circuitry is optional and can be used to assist the three sequences below.

39.4.2.1 Auto-Reset

When the HAAR is set to one, the host automatically starts a reset after a connect event. This shortcuts the normal process where software is notified of the connect event and starts the reset. Software receives notification of the connect event, but should not write the reset bit when the HAAR is set. Software is notified again after the reset is complete via the enable change bit in the PORTSC register that causes a port change interrupt.

This assist ensures the OTG parameter `TB_ACON_BSE0_MAX = 1ms` is met.

39.4.2.2 Data-Pulse

Writing a one to HADP starts a data pulse of approximately 7ms in duration and then automatically cease the data pulsing. During the data pulse, the DP is set and then cleared. This automation relieves software from accurately controlling the data-pulse duration. During the data pulse, the HCD can poll to see that the HADP and DP bit have returned low to recognize the completion or simply launch the data pulse and wait to see if a VBUS Valid interrupt occurs when the A-side supplies bus power.

This assist ensures data pulsing meets the OTG requirement of $> 5\text{ms}$ and $< 10\text{ms}$.

39.4.2.3 B-Disconnect to A-Connect

During HNP, the B-disconnect occurs from the OTG A_suspend state and within 3 ms, the A-device must enable the pullup on the DP leg in the A-peripheral state. When HABA is set, the Host Controller port is in suspend mode, and the device disconnects, then this hardware assist begins.

4. Reset the OTG core.
5. Write the OTG core into device mode.
6. Write the device run bit to a '1' and enable necessary interrupts including:
 - USB Reset Enable (URE) ; enables interrupt on usb bus reset to device
 - Sleep Enable (SLE) ; enables interrupt on device suspend
 - Port Change Detect Enable (PCE) ; enables interrupt on device connect

When software has enabled this hardware assist, it must not interfere during the transition and should not write any register in the core until it gets an interrupt from the device controller signifying that a reset interrupt has occurred or at least first verify that the core has entered device mode. HCD/DCD must not activate the core soft reset at any time since this action is performed by hardware. During the transition, the software may see an interrupt from the disconnect and/or other spurious interrupts (i.e. SOF/etc.) that may or may not cascade and may be cleared by the soft reset depending on the software response time.

After the core has entered device mode by the hardware assist, the DCD must ensure that the `ENDPTLISTADDR` is programmed properly before the host sends a setup packet. Since the end of the reset duration, which may be initiated quickly (a few microseconds) after connect, requires at a minimum 50 ms, this is the time for which the DCD must be ready to accept setup packets after having received notification that the reset has been detected or simply that the OTG is in device mode which ever occurs first.

In the case where the A-peripheral fails to see a reset after the controller enters device mode and engages the DP-pullup, the device controller interrupt the DCD signifying that a suspend has occurred.

This assist ensures the parameter `TA_BDIS_ACON_MAX = 3ms` is met.

39.5 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the enhanced host controller (hardware). The data structure definitions in this section support a 32-bit memory buffer address space. The interface consists of a periodic schedule, periodic frame list, asynchronous schedule, isochronous transaction descriptors, split-transaction isochronous transfer descriptors, queue heads, and queue element transfer descriptors.

The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using isochronous transaction descriptors. Isochronous split-transaction data streams are managed with split-transaction isochronous transfer descriptors. All interrupt, control, and bulk data streams are managed via queue heads and queue element transfer descriptors. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Software must ensure that no interface data structure reachable by the EHCI host controller spans a 4K-page boundary.

The data structures defined in this section are (from the host controller's perspective) a mix of read-only and read/writable fields. The host controller must preserve the read-only fields on all data structure writes.

39.5.1 Periodic Frame List

Figure 39-45 shows the organization of the periodic schedule. This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the `PERIODICLISTBASE` address register and the `FRINDEX` register. The periodic schedule is based on an array of pointers called the Periodic Frame List. The `PERIODICLISTBASE` address register is combined with the `FRINDEX` register to produce a memory pointer into the frame list. The periodic frame list implements a sliding window of work over time.

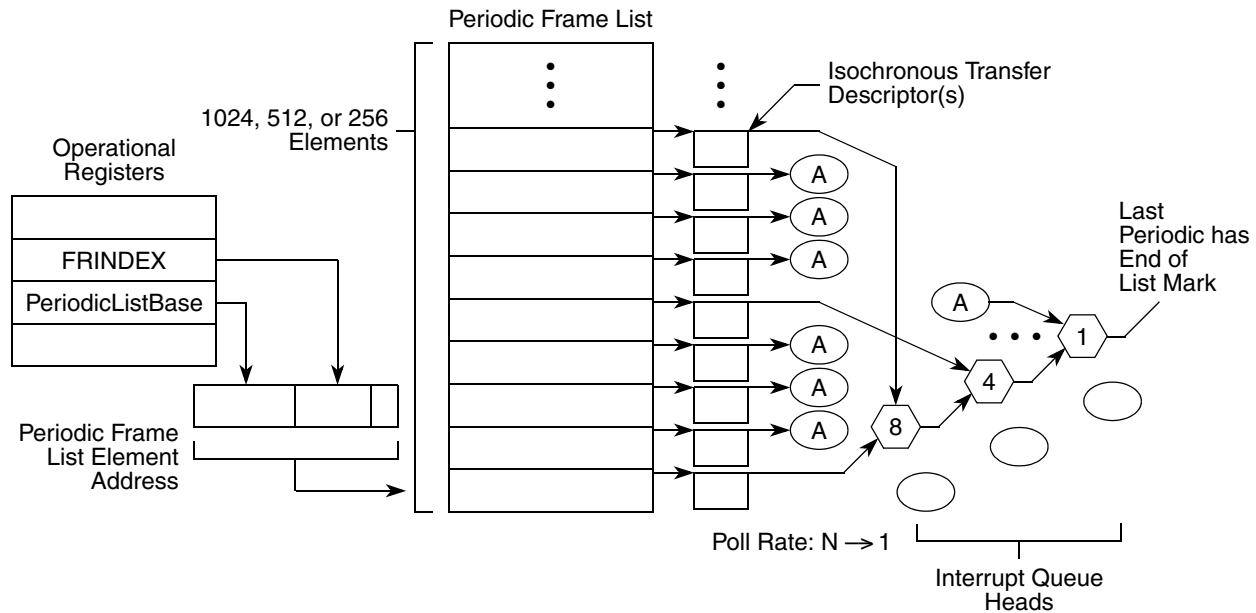


Figure 39-45. Periodic Schedule Organization

Split transaction interrupt, bulk and control are also managed using queue heads and queue element transfer descriptors.

The periodic frame list is a 4K-page aligned array of frame list link pointers. The length of the frame list may be programmable. The programmability of the periodic frame list is exported to system software via the HCCPARAMS register. If non-programmable, the length is 1024 elements. If programmable, system software can select the length as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into frame list size field in the USBCMD register.

Frame list link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on doubleword boundaries within the frame list.

Figure 39-46 shows the format for the frame list link pointer.

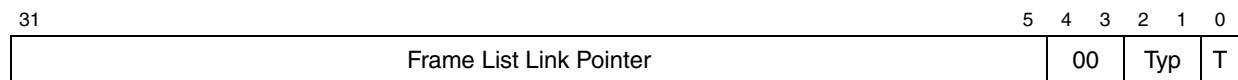


Figure 39-46. Frame List Link Pointer Format

Frame list link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least significant bits in a frame list pointer key the host controller as to the type of object the pointer is referencing.

The least significant bit is the T-Bit (bit 0). When this bit is set, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field is used to indicate the exact type of data structure referenced by this pointer. The value encodings for the Typ field are given in Table 39-45.

Table 39-45. Typ Field Encodings

Typ	Description
00	Isochronous Transfer Descriptor
01	Queue Head
10	Split Transaction Isochronous Transfer Descriptor
11	Frame Span Traversal Node.

39.5.2 Asynchronous List Queue Head Pointer

The asynchronous transfer list (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers only use this list when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.

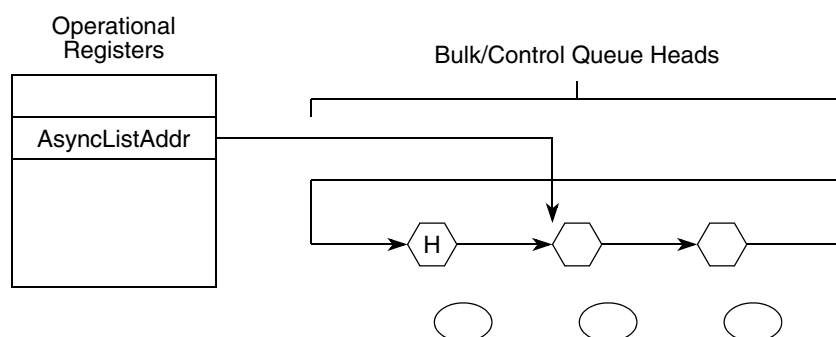


Figure 39-47. Asynchronous Schedule Organization

The asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is a pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

39.5.3 Isochronous (High-Speed) Transfer Descriptor (iTID)

The format of an isochronous transfer descriptor is illustrated in [Figure 39-48](#). This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. Isochronous TDs must be aligned on a 32-byte boundary.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset						
Next Link Pointer																												00	Typ	T	0x00							
Status ¹		Transaction 0 Length ¹										IOC	PG ²	Transaction 0 Offset ²															0x04									
Status ¹		Transaction 1 Length ¹										IOC	PG ²	Transaction 1 Offset ²															0x08									
Status ¹		Transaction 2 Length ¹										IOC	PG ²	Transaction 2 Offset ²															0x0C									
Status ¹		Transaction 3 Length ¹										IOC	PG ²	Transaction 3 Offset ²															0x10									
Status ¹		Transaction 4 Length ¹										IOC	PG ²	Transaction 4 Offset ²															0x14									
Status ¹		Transaction 5 Length ¹										IOC	PG ²	Transaction 5 Offset ²															0x18									
Status ¹		Transaction 6 Length ¹										IOC	PG ²	Transaction 6 Offset ²															0x1C									
Status ¹		Transaction 7 Length ¹										IOC	PG ²	Transaction 7 Offset ²															0x20									
Buffer Pointer (Page 0)																				EndPt			R	Device Address							0x24							
Buffer Pointer (Page 1)																				I/O	Maximum Packet Size																	0x28
Buffer Pointer (Page 2)																				Reserved													Mult	0x2C				
Buffer Pointer (Page 3)																				Reserved													0x30					
Buffer Pointer (Page 4)																				Reserved													0x34					
Buffer Pointer (Page 5)																				Reserved													0x38					
Buffer Pointer (Page 6)																				Reserved													0x3C					

Figure 39-48. Isochronous Transaction Descriptor (iTD)

¹ Host controller read/write; all others read-only.

² These fields may be modified by the host controller if the I/O field indicates an OUT.

39.5.3.1 Next Link Pointer

The first doubleword of an iTD is a pointer to the next schedule data structure.

Table 39-46. Next Schedule Element Pointer

Field	Description
31–5 Link Pointer	These bits correspond to memory address signals [31:5], respectively. This field points to another isochronous transaction descriptor (iTD/siTD) or queue head (QH).
4,3	Reserved. These bits are reserved and their value has no effect on operation. Software should initialize this field to 0.
2,1 Typ	This field indicates to the host controller whether the item referenced is an iTD, siTD, or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0 T	Terminate 1 Link Pointer field is not valid. 0 Link Pointer field is valid.

39.5.3.2 iTD Transaction Status and Control List

doublewords one through eight are eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and transaction n offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description plus the endpoint information contained in the first three doublewords of the buffer page pointer list to execute a transaction on the USB.

Table 39-47. iTD Transaction Status and Control

Field	Description
31–28 Status	This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding: 31 Active. Set by software to enable the execution of an isochronous transaction by the host controller. When the transaction associated with this descriptor is completed, the host controller sets this bit to 0 indicating that a transaction for this element should not be executed when it is next encountered in the schedule. 30 Data Buffer Error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overflow) or is unable to supply data fast enough during transmission (underflow). If an overflow condition occurs, no action is necessary. 29 Babble Detected. Set by the host controller during status update when babble is detected during the transaction generated by this descriptor. 28 Transaction Error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit may only be set for isochronous IN transactions.
27–16 Transacti on n Length	For an OUT, this field is the number of data bytes the host controller sends during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back this field with the number of bytes the host expects to receive. The value in this register is the actual byte count (for example, 0 zero length data, 1 one byte, 2 two bytes, etc.). The maximum value this field may contain is 0xC00 (3072).
15 IOC	Interrupt on complete. If this bit is set, it specifies that when this transaction completes, the host controller should issue an interrupt at the next interrupt threshold.
14–12 PG	These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11–0 Transacti on n Offset	This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

39.5.3.3 iTD Buffer Page Pointer List (Plus)

doublewords 9-15 of an isochronous transaction descriptor are nominally page pointers (4K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven

pointers allow for $3 \text{ (transactions)} \times 1024 \text{ (maximum packet size)} \times 8 \text{ (transaction records)} = 24576$ bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Because each pointer is a 4K aligned page pointer, the least significant 12 bits in several of the page pointers are for other purposes.

Table 39-48. Buffer Pointer Page 0 (Plus)

Field	Description
31:12 Buffer Pointer (Page 0)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:8 EndPt	This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	Reserved. Reserved for future use and should be initialized by software to 0.
6:0 Device Address	This field selects the specific device serving as the data source or sink.

Table 39-49. iTD Buffer Pointer Page 1 (Plus)

Field	Description
31:12 Buffer Pointer (Page 1)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11 I/O	Direction (I/O). This field encodes whether the high-speed transaction should use an IN or OUT PID. 0 OUT 1 IN
10:0 Maximum Packet Size	This directly corresponds to the maximum packet size of the associated endpoint (<i>wMaxPacketSize</i>). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (.for example, per micro-frame). This field is used with the Multi field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (400h). Any larger value yields undefined results.

Table 39-50. Buffer Pointer Page 2 (Plus)

Field	Description
31:12 Buffer Pointer (Page 2)	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].

Table 39-50. Buffer Pointer Page 2 (Plus) (continued)

11:2	Reserved. This bit reserved for future use and should be cleared.
1:0 Mult	This field is used to indicate to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame). The valid values are: 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame

Table 39-51. Buffer Pointer Page 3-6

Field	Description
31:12 Buffer Pointer	This is a 4K aligned pointer to physical memory. Corresponds to memory address bits [31:12].
11:2	Reserved. These bits reserved for future use and should be cleared.

39.5.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next Link Pointer																												00	Typ	T	0x00	
I/O		Port Number						0	Hub Address						0000				EndPt				0	Device Address						0x04		
0000_0000_0000_00000												μFrame C-mask						μFrame S-mask										0x08				
IO C		P ¹	0000				Total Bytes to Transfer ¹						μFrame C-prog-mask ¹						Status ¹						0x0C							
Buffer Pointer (Page 0)														Current Offset ¹														0x10				
Buffer Pointer (Page 1)														000_0000						TP ¹		T-count ¹				0x14						
Back Pointer																												0000		T	0x18	

Figure 39-49. Split-Transaction Isochronous Transaction Descriptor (siTD)

¹ Host controller read/write; all others read-only.

39.5.4.1 Next Link Pointer

Doubleword0 of a siTD is a pointer to the next schedule data structure.

Table 39-52. Next Link Pointer

Bit	Description
31:5 Next Link Pointer	This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4:3	Reserved. These bits must be written as zeros.
2:1 Typ	This field indicates to the host controller whether the item referenced is an iTD/siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0 T	Terminate. 0 Link Pointer is valid. 1 Link Pointer field is not valid.

39.5.4.2 siTD Endpoint Capabilities/Characteristics

doublewords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent companion controller, and micro-frame scheduling control.

Table 39-53. Endpoint and Transaction Translator Characteristics

Bit	Description
31 I/O	Direction (I/O). This field encodes whether the full-speed transaction should be an IN or OUT. 0 OUT 1 IN
30:24 Port Number	This field is the port number of the recipient transaction translator.
23	Reserved. Bit reserved and should be cleared.
22:16 Hub Address	This field holds the device address of the companion controllers' hub.
15:12	Reserved. Field reserved and should be cleared.
11:8 EndPt	Endpoint Number. This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	Reserved. Bit is reserved for future use. It should be cleared.
6:0 Device Address	This field selects the specific device serving as the data source or sink.

Table 39-54. Micro-frame Schedule Control

Bit	Description
31:16	Reserved. This field reserved for future use. It should be cleared.
15:8 μFrame C-mask	Split Completion Mask. This field (along with the active and SplitX- state fields in the status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame C-Mask field is a one, then this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7:0 μFrame S-mask	Split Start Mask. This field (along with the active and SplitX-state fields in the status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

39.5.4.3 siTD Transfer State

Doublewords 3-6 are used to manage the state of the transfer.

Table 39-55. siTD Transfer Status and Control

Bit	Description
31 IOC	Interrupt On Complete 0 Do not interrupt when transaction is complete. 1 Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed, it asserts a hardware interrupt at the next interrupt threshold.
30 P	Page Select. Used to indicate which data page pointer should be concatenated with the CurrentOffset field to construct a data buffer pointer 0 selects Page 0 pointer 1 selects Page 1 pointer The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a 1 to a 0).
29:26	Reserved. This field reserved for future use and should be cleared.
25:16 Total Bytes to Transfer	Software initializes this field to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)
15:8 μFrame C-prog-m ask	Split complete progress mask. The host controller uses this field to record which split-completes have been executed.

Table 39-55. siTD Transfer Status and Control (continued)

Bit	Description	
7:0 Status	This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:	
	Status Bit	Definition
	7	Active. Set by software to enable the execution of an isochronous split transaction by the host controller.
	6	ERR. Set by the host controller when an ERR response is received from the Companion Controller.
	5	Data Buffer Error. Set by the host controller during status update to indicate the host controller is unable to keep up with the reception of incoming data (overflow) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the host controller transmits an incorrect CRC (thus invalidating the data at the endpoint). If an overflow condition occurs, no action is necessary.
	4	Babble Detected. Set by the host controller during status update when babble is detected during the transaction generated by this descriptor.
	3	Transaction Error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit is only set for IN transactions.
	2	Missed Micro-Frame. The host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction.
	1	Split Transaction State (SplitXstate). The bit encodings are: 0 Do Start Split. This value directs the host controller to issue a start split transaction to the endpoint when a match is encountered in the S-mask. 1 Do Complete Split. This value directs the host controller to issue a complete split transaction to the endpoint when a match is encountered in the C-mask.
	0	Reserved. Bit reserved for future use and should be cleared.

39.5.4.4 siTD Buffer Pointer List (Plus)

Doublewords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most significant 20 bits of each doubleword in this section are the 4K (page) aligned buffer pointers. The least significant 12 bits of each doubleword is an additional transfer state.

Table 39-56. siTD Buffer Pointer Page 0 (Plus)

Bit	Description
31:12 Buffer Pointer (Page 0)	Bits [31:12] is a 4K page-aligned, physical memory address. These bits correspond to physical address bits [31:12] respectively. The field P specifies the current active pointer
11:0 Current Offset	The 12 least significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).

Table 39-57. siTD Buffer Pointer Page 1 (Plus)

Bit	Description
31:12 Buffer Pointer (Page 1)	Bits [31:12] is a 4K page-aligned, physical memory address. These bits correspond to physical address bits [31:12] respectively. The field P specifies the current active pointer
11:5	Reserved.
4:3 TP	Transaction position. This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: 00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 11 End. This is the last payload for a full-speed OUT transaction that was larger than 188 bytes.
2:0 T-Count	Transaction count. Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

39.5.4.5 siTD Back Link Pointer

Doubleword 6 of a siTD is simply another schedule link pointer. This pointer is always zero or references an siTD. This pointer cannot reference any other schedule data structure.

Table 39-58. siTD Back Link Pointer

Bit	Description
31:5 Back Pointer	This field is a physical memory pointer to a siTD.
4:1	Reserved. This field is reserved for future use. It should be cleared.
0 T	Terminate 0 siTD Back Pointer field is valid 1 siTD Back Pointer field is not valid

39.5.5 Queue Element Transfer Descriptor (qTD)

This data structure is used only with a queue head. This data structure is for one or more USB transactions. This data structure transfers up to 20480 (5×4096) bytes. The structure contains two structure pointers used for queue advancement, a doubleword of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Next qTD Pointer																												0000	T	0x00		
Alternate Next qTD Pointer																												0000	T	0x04		
dt ¹	Total Bytes to Transfer ¹														IO C	C_Page ¹	Cerr ¹	PID Code	Status ¹								0x08					
Buffer Pointer (Page 0)												Current Offset ¹																0x0C				
Buffer Pointer (Page 1)												0000_0000_0000																0x10				
Buffer Pointer (Page 2)												0000_0000_0000																0x14				
Buffer Pointer (Page 3)												0000_0000_0000																0x18				
Buffer Pointer (Page 4)												0000_0000_0000																0x1C				

Figure 39-50. Queue Element Transfer Descriptor (qTD)

¹ Host controller read/write; all others read-only.

Queue element transfer descriptors must be aligned on 32-byte boundaries.

39.5.5.1 Next qTD Pointer

The first doubleword of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 39-59. qTD Next Element Transfer Pointer (doubleword 0)

Bit	Description
31:5 Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed. The field corresponds to memory address signals[31:5], respectively.
4:1	Reserved. These bits are reserved and their value has no effect on operation.
0 T	Terminate. This bit indicates to the Host Controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid Transfer Element Descriptor). 1 Pointer is invalid.

39.5.5.2 Alternate Next qTD Pointer

The second doubleword of a queue element transfer descriptor supports hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit, the host controller always uses this pointer when the current qTD is retired due to short packet.

Table 39-60. qTD Alternate Next Element Transfer Pointer (Doubleword 1)

Bit	Description
31:5 Alternate Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively.

Table 39-60. qTD Alternate Next Element Transfer Pointer (Doubleword 1) (continued)

Bit	Description
4:1	Reserved. These bits are reserved and their value has no effect on operation.
0 T	Terminate. This bit indicates to the Host Controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid Transfer Element Descriptor). 1 Pointer is invalid.

39.5.5.3 qTD Token

The third doubleword of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head). Some of the field descriptions in [Table 39-61](#) reference fields defined in the queue head. See [Section 39.5.6, “Queue Head,”](#) for more information on these fields.

Table 39-61. qTD Token (doubleword 2)

Bit	Description
31 Data Toggle	This is the data toggle sequence bit. The use of this bit depends on the setting of the data toggle control bit in the queue head.
30:16 Total Bytes to Transfer	Total Bytes to Transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction on the successful completion of the transaction. The maximum value software may store in this field is $5 \times 4K$ (0x5000). This is the maximum number of bytes five page pointers can access. If the value of this field is 0 when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that total bytes to transfer be an even multiple of QH [Maximum Packet Length]. If software builds such a transfer descriptor for an OUT transfer, the last transaction is always less than QH [Maximum Packet Length]. Although it is possible to create a transfer up to 20K this assumes the page is zero. When the offset cannot be predetermined, crossing past the fifth page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K(0x4000).
15 IOC	Interrupt On Complete. If this bit is set, it specifies that when this qTD is completed, the host controller should issue an interrupt at the next interrupt threshold.
14:12 C_Page	Current Page. This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0x0 to 0x4. The host controller is not required to write this field back when the qTD is retired.

Table 39-61. qTD Token (doubleword 2) (continued)

Bit	Description												
11:10 Cerr	<p>Error Counter. This field is a 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during set-up, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt is generated if the USB error interrupt enable bit in the USBINTR register is set. If the host controller driver (HCD) software programs this field to zero during set-up, the host controller does not count errors for this qTD and there is no limit on the retries of this qTD. Write-backs of intermediate execution state are to the queue head overlay area, not the qTD.</p> <table> <tr> <th>Error</th><th>Decrement Counter</th></tr> <tr> <td>Transaction Error</td><td>Yes</td></tr> <tr> <td>Data Buffer Error</td><td>No. Data buffer errors are host problems. They don't count against the device's retries. Software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.</td></tr> <tr> <td>Stalled</td><td>No. Detection of babble or stall automatically halts the queue head. Count is not decremented</td></tr> <tr> <td>Babble Detected</td><td>No. Detection of babble or stall automatically halts the queue head. Count is not decremented</td></tr> <tr> <td>No Error</td><td>No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT), a bus transaction completes, and the host controller does not detect a transaction error, the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.</td></tr> </table>	Error	Decrement Counter	Transaction Error	Yes	Data Buffer Error	No. Data buffer errors are host problems. They don't count against the device's retries. Software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.	Stalled	No. Detection of babble or stall automatically halts the queue head. Count is not decremented	Babble Detected	No. Detection of babble or stall automatically halts the queue head. Count is not decremented	No Error	No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT), a bus transaction completes, and the host controller does not detect a transaction error, the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.
Error	Decrement Counter												
Transaction Error	Yes												
Data Buffer Error	No. Data buffer errors are host problems. They don't count against the device's retries. Software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.												
Stalled	No. Detection of babble or stall automatically halts the queue head. Count is not decremented												
Babble Detected	No. Detection of babble or stall automatically halts the queue head. Count is not decremented												
No Error	No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT), a bus transaction completes, and the host controller does not detect a transaction error, the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.												
9:8 PID Code	<p>This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:</p> <ul style="list-style-type: none"> 00 OUT Token generates token (E1H) 01 IN Token generates token (69H) 10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example. μFrame S-mask field in the queue head is non-zero.) 11 Reserved 												

Table 39-61. qTD Token (doubleword 2) (continued)

Bit	Description
7:0 Status	The host controller uses this field to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:
Bit	Status Field Description
7	Active. Set by software to enable the execution of transactions by the host controller.
6	Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time a transaction results in the halted bit being set, the active bit is also cleared.
5	Data Buffer Error. Set by the host controller during status update to indicate the host controller is unable to keep up with the reception of incoming data (overflow) or is unable to supply data fast enough during transmission (under run). If an overflow condition occurs, the host controller forces a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, it remains a one for the duration of the transfer.
4	Babble Detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the halted bit to a one. Because babble is considered a fatal error for the transfer, setting the halted bit to a one ensures no more transactions occur because of this descriptor.
3	Transaction Error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, it remains a one for the duration of the transfer.
2	Missed Micro-Frame. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detects a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, it remains a one for the duration of the transfer.
1	Split Transaction State (SplitXstate). This bit is ignored by the host controller unless the QH[EPS] field indicates a full- or low-speed endpoint. When configured as a full- or low-speed device, the host controller uses this bit to track the state of the split- transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are: 0 Do Start Split. This value directs the host controller to issue a start split transaction to the endpoint. 1 Do Complete Split. This value directs the host controller to issue a complete split transaction to the endpoint.
0	Ping State (P)/ERR. If the QH[EPS] field indicates a high-speed device and the PID Code indicates an OUT endpoint, this is the state bit for the ping protocol. The bit encodings are: 0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint. If the QH[EPS] field does not indicate a high-speed device, this field is used as an error indicator bit. It is set by the host controller when a periodic split-transaction receives an ERR handshake.

39.5.5.4 qTD Buffer Page Pointer List

The last five doublewords of a queue element transfer descriptor is an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes Current Offset field to the starting offset into the current page, where current page is selected via the value in the C_Page field.

Table 39-62. qTD Buffer Pointer

Bit	Name	Description
31:12	Buffer Pointer (page <i>n</i>)	Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11:0	Current Offset (Page 0)/— (Pages 1-4)	This field is reserved in all pointers except the first one (that is, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. Software should ensure the reserved fields are initialized to zeros.

39.5.6 Queue Head

Figure 39-51 shows the queue head structure.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Queue Head Horizontal Link Pointer																											00		Typ		T	0x00
RL			C	Maximum Packet Length										H	dtc	EPS		EndPt			I	Device Address										0x04 ¹
Mult		Port Number					Hub Addr					μFrame C-mask										μFrame S-mask										0x08 ¹
Current qTD Pointer ²																											00000		0x0C			
Next qTD Pointer ²																											0000		T ²	0x10 ³		
Alternate Next qTD Pointer ²																											NakCnt ²		T ²	0x14 ^{3,4}		
dt ¹	Total Bytes to Transfer ²											IOC ₂		C_Page ²		Cerr ²		PID Code ²		Status ²										0x18 ^{3,4}		
Buffer Pointer (Page 0) ²															Current Offset ²												0x1C ^{3,4}					
Buffer Pointer (Page 1) ²															0000		C-prog-mask ²										0x20 ^{3,4}					
Buffer Pointer (Page 2) ²															S-bytes ²										FrameTag ²		0x24 ^{3,4}					
Buffer Pointer (Page 3) ²															0000_0000_0000												0x28 ³					
Buffer Pointer (Page 4) ²															0000_0000_0000												0x2C ³					

¹ Offsets 0x04 through 0x0B contain the static endpoint state.

² Host controller read/write; all others read-only.

³ Offsets 0x10 through 0x2F contain the transfer overlay.

⁴ Offsets 0x14 through 0x27 contain the transfer results.

Figure 39-51. Queue Head Layout

39.5.6.1 Queue Head Horizontal Link Pointer

The first doubleword of a queue head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

Table 39-63. Queue Head doubleword 0

Field	Description
31:5 QHLP	Queue Head Horizontal Link Pointer. This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively.
4:3	Reserved. These bits must be written as zeros.

Table 39-63. Queue Head doubleword 0 (continued)

Field	Description
2:1 Typ	This field indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0 T	Terminate. 1 Last QH (pointer is invalid). 0 Pointer is valid. If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. The host controller ignores this bit when the queue head is in the asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.

39.5.6.2 Endpoint Capabilities/Characteristics

The second and third doublewords of a queue head specify static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint Characteristics. These are the USB endpoint characteristics including addressing, maximum packet size, and endpoint speed.
- Endpoint Capabilities. These are adjustable parameters of the endpoint. They affect how the endpoint data stream is managed by the host controller.
- Split Transaction Characteristics. This data structure manages full- and low-speed data streams for bulk, control, and interrupt via split transactions to USB2.0 hub transaction translator. There are additional fields for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 39-64. Endpoint Characteristics: Queue Head doubleword 1

Field	Description
31:28 RL	Nak Count Reload. This field contains a value used by the host controller to reload the Nak counter field.
27 C	Control Endpoint Flag. If the QH[EPS] field indicates the endpoint is not a high-speed device and the endpoint is a control endpoint, software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26:16 Maximum Packet Length	This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15 H	Head of reclamation list flag. This bit is set by system software to mark a queue head as being the head of the reclamation list.

Table 39-64. Endpoint Characteristics: Queue Head doubleword 1 (continued)

Field	Description
14 DTC	Data Toggle Control (DTC). This bit specifies where the host controller should get the initial data toggle on an overlay transition. 0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1 Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.
13:12 EPS	Endpoint Speed. This is the speed of the associated endpoint. 00 Full-Speed (12Mbps) 01 Low-Speed (1.5Mbps) 10 High-Speed (480 Mb/s) 11 Reserved This field must not be modified by the host controller.
11:8 EndP I	Endpoint Number. This 4-bit field selects the particular endpoint number on the device serving as the data source or sink.
7	Inactivate on next transaction. This bit is used by system software to request the host controller set the active bit to zero. This field is only valid when the queue head is in the periodic schedule and the EPS field indicates a full or low-speed endpoint. Setting this bit to a one when the queue head is in the asynchronous schedule or the EPS field indicates a high-speed device yields undefined results.
6:0 Device Address	This field selects the specific device serving as the data source or sink.

Table 39-65. Endpoint Capabilities: Queue Head doubleword 2

Field	Description
31:30 Mult	High-Bandwidth Pipe Multiplier. This field is a multiplier used to key the host controller to the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
29:23 Port Number	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.
22:16 Hub Addr	This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.

Table 39-65. Endpoint Capabilities: Queue Head doubleword 2 (continued)

Field	Description
15:8 μFrame C-mask	This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the μFrame C- mask field is a one, this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7:0 μFrame S-mask	Interrupt Schedule Mask. This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the μFrame S-mask field has a one at the indexed bit position, this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types such as Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list, has undefined results.

39.5.6.3 Transfer Overlay

The nine doublewords in this area represent a transaction working space for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, it follows the queue head horizontal link pointer to the next queue head. The host controller never follows the next transfer queue element or alternate queue element pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The doubleword of a queue head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 39-66. Current qTD Link Pointer

Field	Description
31:5 Current qTD Pointer	Current Element Transaction Descriptor Link Pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively.
4:0	Reserved. These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The doublewords 4-11 of a queue head are the transaction overlay area. This area has the same base structure as a queue element transfer descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

Table 39-67. Host-Controller Rules for Bits in Overlay (doublewords 5, 6, 8 and 9)

doubleword	QH Offset	Bit	Description
5	0x14	4:1 NakCnt	Nak counter—RW. This field is a counter the host controller decrements when a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an asynchronous list restart condition). It is also loaded from RL during an overlay.
6	0x18	31 DT	The data toggle control controls whether the host controller preserves this bit when an overlay operation is performed.
6	0x18	15 IOC	Interrupt on complete. The IOC control bit is always inherited from the source qTD when the overlay operation is performed.
6	0x18	11:10 Cerr	Error counter. This two-bit field is copied from the qTD during the overlay and written back during queue advancement.
6	0x18	0 Status	Ping state (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	0x20	7:0 C-prog-mask	Split-transaction complete-split progress. This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	0x24	11:5 S-bytes	Software must ensure that the S-bytes field in a qTD is zero before activating the qTD. This field is used to keep track of the number of bytes sent or received during an IN or OUT split transaction.
9	0x24	4:0 FrameTag	Split-transaction frame tag. This field is initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.

39.5.7 Periodic Frame Span Traversal Node (FSTN)

This data structure is only for managing full- and low-speed transactions that span a host-frame boundary. Software must not use an FSTN in the asynchronous schedule. An FSTN in the asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation below 0x0096. FSTNs were not defined for EHCI implementations before revision 0.96 of the EHCI specification and their use may yield undefined results.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	offset
Normal Path Link Pointer																										00	Typ	T	0x00			
Back Path Link Pointer																										00	Typ	T	0x04			

Figure 39-52. Frame Span Traversal Node Structure

39.5.7.1 FTSN Normal Path Pointer

The first doubleword of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

Table 39-68. FTSN Normal Path Pointer

Bit	Description
31:5 NPLP	Normal Path Link Pointer. This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4:3	Reserved. These bits must be written as 0s.
2:1 Typ	This field indicates to the host controller whether the item referenced is a iTD/siTD, a QH, or an FSTN. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (Frame Span Traversal Node)
0 T	Terminate. 0 Link Pointer is valid. 1 Link Pointer field is not valid.

39.5.7.2 FSTN Back Path Link Pointer

The second doubleword of an FTSN node contains a link pointer to a queue head. If the T-bit in this pointer is a zero, then this FSTN is a save-place indicator. Its Typ field must be set by software to indicate the target data structure is a queue head. If the T-bit in this pointer is set, this FSTN is the restore indicator. When the T-bit is a one, the host controller ignores the Typ field.

Table 39-69. FSTN Back Path Link Pointer

Bit	Description
31:5 BPLP	Back Path Link Pointer. This field contains the address of a queue head. This field corresponds to memory address signals [31:5], respectively.
4:3	Reserved. These bits must be written as 0s.
2:1 Typ	Software must ensure this field is set to indicate the target data structure is a queue head (01). Any other value in this field yields undefined results.
0 T	Terminate. 0 Link Pointer is valid (that is, the host controller may use bits [31:5] [in combination with the CTRLDSSEGMENT register if applicable] as a valid memory address). This value also indicates that this FSTN is a save-place indicator. 1 Link Pointer field is not valid (that is, the host controller must not use bits [31:5] [in combination with the CTRLDSSEGMENT register if applicable] as a valid memory address). This value also indicates that this FSTN is a restore indicator.

39.6 Host Operational Model

The general operational model for the USB modules in host mode is defined by the enhanced host controller interface (EHCI) specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface between system software and host controller hardware. Information concerning the initialization of the USB modules is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

39.6.1 Host Controller Initialization

After initial power-on or HCRreset (hardware or via HCRreset bit in the USBCMD register), all of the operational registers are at their default values, as illustrated in [Table 39-70](#). After a hardware reset, only the operational registers not contained in the auxiliary power well is at their default values.

Table 39-70. Default Values of Operational Register Space

Operational Register	Default Value (after Reset)
USBCMD	0x0008_0000 (0x0008_0B00 if Asynchronous Schedule Park Capability is set)
USBSTS	0x0000_1000
USBINTR	0x0000_0000
FRINDEX	0x0000_0000
CTRLDSSEGMENT	0x0000_0000
PERIODICLISTBASE	Undefined
ASYNCLISTADDR	Undefined
CONFIGFLAG	0x0000_0000
PORTSC	0x0000_2000 (w/PPC set); 0x0000_3000 (w/PPC cleared)

To initialize the host controller, software should perform the following steps:

1. Optionally set streaming disable in the USBMODE register.
2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSCx register if using a non-ULPI PHY.
4. Set the USB_EN bit in the CONTROL register.
5. Program the CTRLDSSEGMENT register with 4-Gigabyte segment where all of the interface data structures are allocated.
6. Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
7. Write the base address of the periodic frame list to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the periodic frame list should have their T-Bits set.
8. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable), and turn the controller by setting the run/stop bit.

At this point, the host controller is running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled high-speed ports, but the schedules have not yet been enabled. The EHCI host controller does not transmit SOFs to enabled full- or low-speed ports.

To communicate with devices via the asynchronous schedule, system software must write the ASYNCLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to the asynchronous schedule enable bit in the USBCMD

register. To communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to the periodic schedule enable bit in the USBCMD register. The schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

39.6.1.1 Port Power

The port power control (PPC) bit in the HCSPARAMS register indicates whether the USB 2.0 host controller has port power control. When the PPC bit is a one, the host controller supports port power switches. Each available switch has an output enable. PPE is controlled based on the state of the combination bits PPC bit, EHCI configured (CF) bit, and individual port power (PP) bits.

39.6.1.2 Reporting Over-Current

Host ports by definition are power providers on USB. Whether the ports are considered high- or low-powered is a platform implementation issue. Each EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

39.6.2 Suspend/Resume

The host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely via software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated or software-initiated resumes are called resume events/actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. Similar to USB 2.0 hubs, when in host mode, the host controller responds to explicit device, resumes signaling, and wakes up the system (if necessary).
- Port connect and disconnect. Sensitivity to these events can be turned on or off by using the port control bits in the PORTSC register. An over-current event does not wake the USB core.

Selective suspend is a feature supported by the PORTSC register. It places specific ports into a suspend mode. This feature is a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the run/stop bit in the USBCMD register to a zero.

When a wake event occurs, the system resumes operation and system software must set the run/stop bit to a one and resume the suspended port.

39.6.2.1 Port Suspend/Resume

System software places the USB into suspend mode by writing a one into the appropriate PORTSC suspend bit. Software must only set the suspend bit when the port is in the enabled state (port enabled bit is a one).

The host controller may evaluate the suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several micro-frames of activity on the port until the host controller evaluates the suspend bit. The host controller must evaluate the suspend bit at least every frame boundary.

System software can initiate a resume on the suspended port by writing a one to the force port resume bit. Software should not attempt to resume a port unless the port reports that it is in the suspended state. If system software sets the force port resume bit when the port is not in the suspended state, the resulting behavior is undefined. To assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates it is suspended (suspend bit is a one) before initiating a port resume via the force port resume bit. When force port resume bit is set, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds), and then clears the force port resume bit. When the host controller receives the write to transition force port resume to zero, it completes the resume sequence as defined in the USB specification, and clears both the force port resume and suspend bits. Software-initiated port resumes do not affect the port change detect bit in the USBSTS register nor do they cause an interrupt if the port change interrupt enable bit in the USBINTR register is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 μ sec. The port's force port resume bit is set and the port change detect bit in the USBSTS register is set. If the port change interrupt enable bit in the USBINTR register is a one, the host controller issues a hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 milliseconds), and then terminates the resume sequence by clearing the force port resume bit in the port. The host controller receives the write of zero to force port resume, terminates the resume sequence, and clears the force port resume and suspend port bits. Software can determine the port is enabled (not suspended) by sampling the PORTSC register and observing that the suspend and force port resume bits are zero. Software must ensure that the host controller is running (HCHalted bit in the USBSTS register is a zero), before terminating a resume by clearing the port's force port Resume bit. If HCHalted is a one when force port resume is cleared, SOFs does not occur down the enabled port and the device returns to suspend mode in a maximum of 10 milliseconds.

[Table 39-71](#) summarizes the wake-up events. When a resume event is detected, the port change detect bit in the USBSTS register is set. If the port change interrupt enable bit is a one in the USBINTR register, the host controller also generates an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the port change detect status bit in the USBSTS register.

Table 39-71. Behavior During Wake-up Events

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	not D0
Port disabled, resume K-State received	No Effect	N/A	N/A
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. Force port resume status bit in PORTSC register is set. Port change detect bit in USBSTS register is set.	¹ , ²	²
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is set. A disconnect is detected.	Depending in the initial port state, the PORTSC connect and enable status bits are cleared, and the connect change status bit is set. Port change detect bit in the USBSTS register is set.	¹ , ²	²
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit is cleared. A disconnect is detected.	Depending on the initial port state, the portsc connect and enable status bits are cleared, and the connect change status bit is set. Port change detect bit in the USBSTS register is set.	¹ , ³	³
Port is not connected and the port's WKCNTNT_E bit is a one. A connect is detected.	PORTSC connect status and connect status change bits are set. Port change detect bit in the USBSTS register is set.	¹ , ²	²
Port is not connected and the port's WKCNTNT_E bit is a zero. A connect is detected.	PORTSC connect status and connect status change bits are set. Port change detect bit in the USBSTS register is set.	¹ , ³	³
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC over-current active, over-current change bits are set. If port enable/disable bit is a one, it is cleared. Port change detect bit in the USBSTS register is set	¹ , ²	²
Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs.	PORTSC over-current active, over-current change bits are set. If port enable/disable bit is a one, it is cleared. Port change detect bit in the USBSTS register is set.	¹ , ³	³

¹ Hardware interrupt issued if port change interrupt enable bit in the USBINTR register is set.

² ME# asserted if enabled. PPME Status must always be set.

³ PME# not asserted.

39.6.3 Schedule Traversal Rules

The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures provide the maximum flexibility required by USB, minimize memory traffic, and hardware/software complexity.

System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the PERIODICLISTBASE register. See [Section 39.2.1.4.6, “Periodic Frame List Base Address Register \(PERIODICLISTBASE\),”](#) for more information. The PERIODICLISTBASE register is the physical memory base address of the periodic frame list. The

periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in [Section 39.5, “Host Data Structures.”](#) In each micro-frame, if the periodic schedule is enabled, then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It only executes from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset references from the PERIODICLISTBASE and the FRINDEX registers (see [Figure 39-53](#)). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transition immediately to traversing the asynchronous schedule. After this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

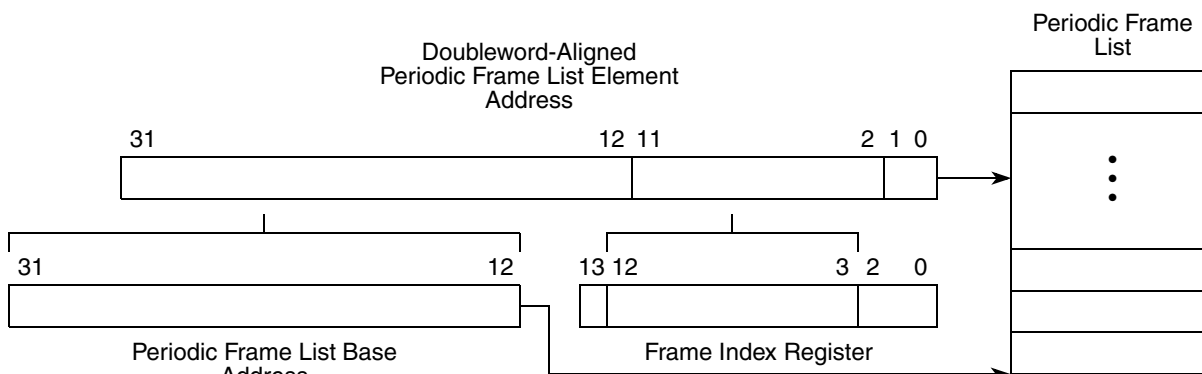


Figure 39-53. Derivation of Pointer into Frame List Array

When the host controller determines it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, as shown in [Figure 39-54](#).

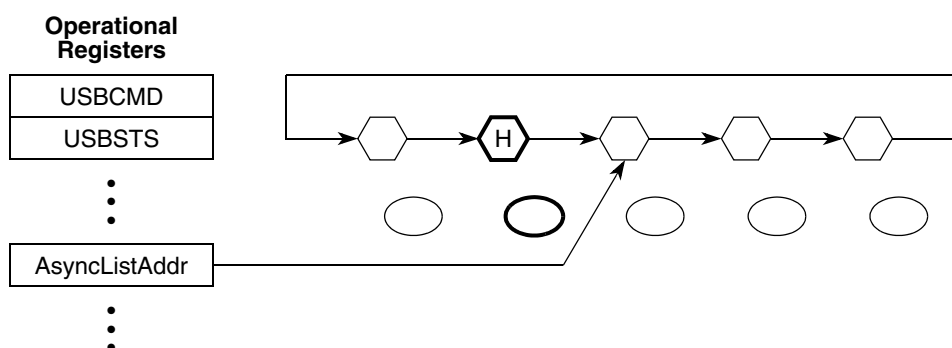


Figure 39-54. General Format of Asynchronous Schedule List

The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule.

39.6.4 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries

The USB specification revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(es) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in [Figure 39-55](#)). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

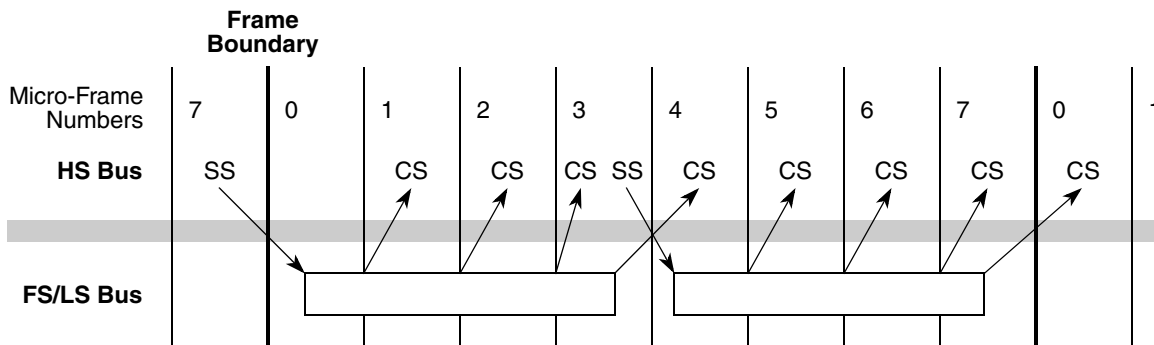


Figure 39-55. Frame Boundary Relationship Between HS Bus and FS/LS Bus

The simple projection, as [Figure 39-55](#) illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. To reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the frame list index register (FRINDEX). Bits FRINDEX[2:0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13:3]. FRINDEX[13:3] and the SOF value are incremented based on FRINDEX[2:0]. The SOF must value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in [Figure 39-56](#). This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface.

[Figure 39-56](#) illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the 1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-millisecond boundaries is called B-Frames.

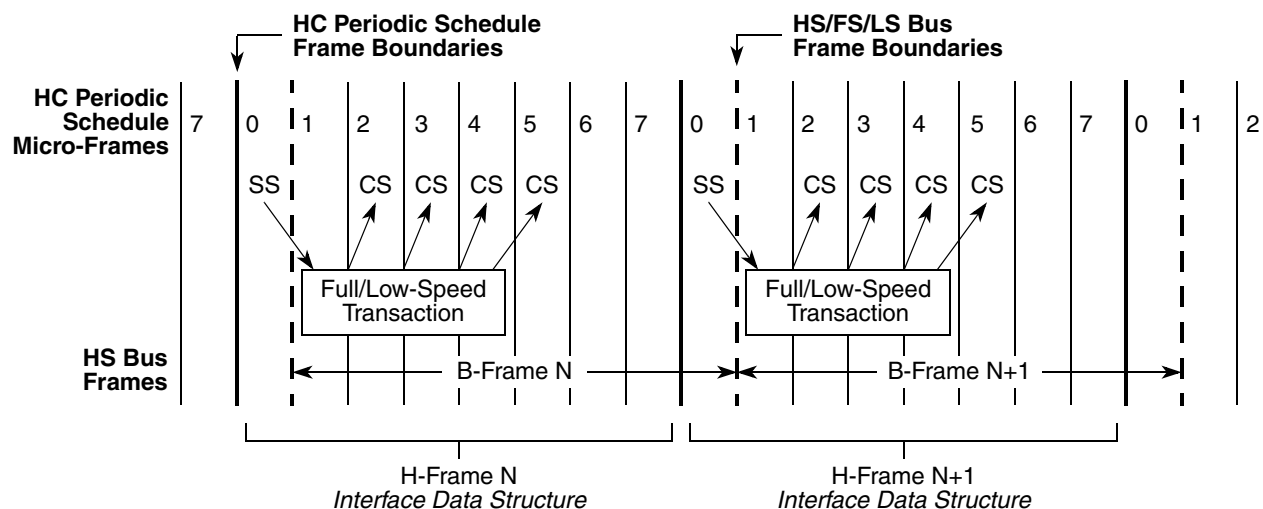


Figure 39-56. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries

H-Frame boundaries for the host controller correspond to increments of `FRINDEX[13:3]`. Micro-frame numbers for the H-Frame are tracked by `FRINDEX[2:0]`. B-Frame boundaries are visible on the high-speed bus via changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (the high-speed bus sees eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 39-56. The host controller's periodic schedule is naturally aligned to H-Frames. Software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline. As described in Section 39.2.1.4.4, “Frame Index Register (FRINDEX),” the SOF value can be implemented as a shadow register (in this example, called SOFV), which lags the `FRINDEX` register bits [13:3] by one micro-frame count. Table 39-72 illustrates the required relationship between the value of `FRINDEX` and the value of SOFV. This lag behavior can be accomplished by incrementing `FRINDEX[13:3]` based on carry-out of the 7 to 0 increment of `FRINDEX[2:0]` and incrementing SOFV based on the transition of 0 to 1 of `FRINDEX[2:0]`.

Software can write to `FRINDEX`. Section 39.2.1.4.4, “Frame Index Register (FRINDEX),” provides the requirements that software should adhere to when writing a new value in `FRINDEX`.

Table 39-72. Operation of FRINDEX and SOFV (SOF Value Register)

Current			Next		
FRINDEX[13:3]	SOFV	FRINDEX[2:0]	FRINDEX[13:3]	SOFV	FRINDEX[2:0]
N	N	111	N+1	N	000
N+1	N	000	N+1	N+1	001
N+1	N+1	001	N+1	N+1	010
N+1	N+1	010	N+1	N+1	011
N+1	N+1	011	N+1	N+1	100
N+1	N+1	100	N+1	N+1	101
N+1	N+1	101	N+1	N+1	110
N+1	N+1	110	N+1	N+1	111

39.6.5 Periodic Schedule

The periodic schedule traversal is enabled or disabled via the periodic schedule enable bit in the USBCMD register. If the periodic schedule enable bit is cleared, the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when the periodic schedule enable bit is a one, the host controller uses the PERIODICLISTBASE register to traverse the periodic schedule. The host controller does not react to modifications to the periodic schedule enable immediately. To eliminate conflicts with split transactions, the host controller evaluates the periodic schedule enable bit only when FRINDEX[2:0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 micro-frame. These work items must be removed from the schedule before the periodic schedule enable bit is cleared. The periodic schedule status bit in the USBSTS register indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by setting (or clearing) the periodic schedule enable bit in the USBCMD register. Software then can poll the periodic schedule status bit to determine when the periodic schedule has made the desired transition. Software must not modify the periodic schedule enable bit unless the value of the periodic schedule enable bit equals that of the periodic schedule status bit.

The periodic schedule manages all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. [Figure 39-57](#) illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the end.

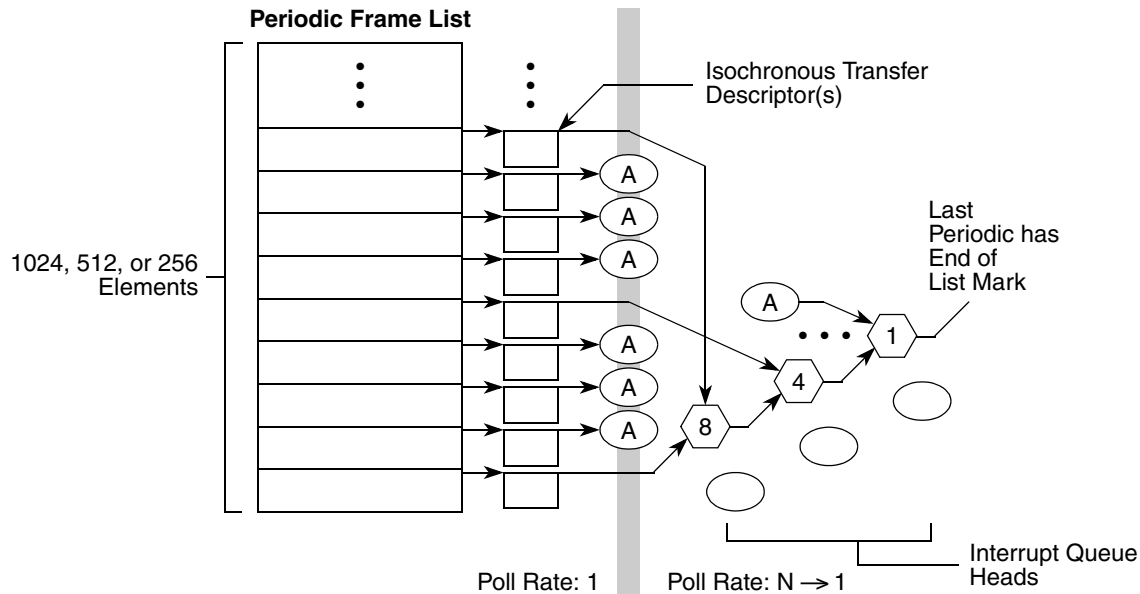


Figure 39-57. Example Periodic Schedule

39.6.6 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Section 29.3.4.3 Isochronous (High-Speed) Transfer Descriptor (iTD). There are four distinct sections to an iTD:

- The first field is the next link pointer. This is for schedule linkage purposes only.
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size, and high-bandwidth multiplier.

39.6.6.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits [12:3] to index into the periodic frame list. This means the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions that map directly to FRINDEX register bits [2:0]. Each iTD can span eight micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits [2:0] to index into the transaction description array. If the active bit in the status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the next pointer to the next schedule data structure.

When the indexed active bit is a one, the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum

packet size, etc.). It also uses the page select (PG) field to index the buffer pointer array, storing the selected buffer pointer, and the next sequential buffer pointer. For example, if PG field is a 0, the host controller stores Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's transaction offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer crosses a page boundary. When this occurs, the host controller replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the maximum packet size field. An iTD supports high-bandwidth pipes via the mult (multiplier) field. When the mult field is 1, 2, or 3, the host controller executes the specified number of maximum packet sized bus transactions for the endpoint in the current micro-frame. In other words, the mult field represents a transaction count for the endpoint in the current micro-frame. If the mult field is zero, the operation of the host controller is undefined. The transfer description services all transactions indicated by the mult field.

For OUT transfers, the value of the transaction n length field represents the total bytes to be sent during the micro-frame. Software must set the mult field to be consistent with transaction n length and maximum packet size. The host controller sends the bytes in maximum packet sized portions. After each transaction, the host controller decrements its local copy of transaction n length by maximum packet size. The number of bytes the host controller sends is always maximum packet size or transaction n length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD, and moves to the next schedule data structure. The maximum sized transaction supported is 3×1024 bytes.

For IN transfers, the host controller issues mult transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use maximum packet size to detect packet babble errors. The host controller keeps the sum of bytes received in the transaction n length field. After all transactions for the endpoint have completed for the micro-frame, transaction n length contains the total bytes received. If the final value of transaction n length is less than the value of maximum packet size, less data than was allowed for was received from the associated endpoint. This short packet condition does not set the USBINT bit in the USBSTS register. The host controller does not detect this condition. If the device sends more than transaction n length or maximum packet size bytes (whichever is less), the host controller sets the babble detected bit and clears the active bit. The host controller is not required to update the iTD field transaction n length in this error scenario. If the mult field is greater than one, the host controller automatically executes the value of mult transactions. The host controller does not execute all mult transactions if:

- The endpoint is an OUT and Transaction n Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame.

39.6.6.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 39-58 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (the periodic frame list and a set of iTDs). On the right of Figure 39-58 is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle of Figure 39-58 are the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left of Figure 39-58 is the host controller's frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

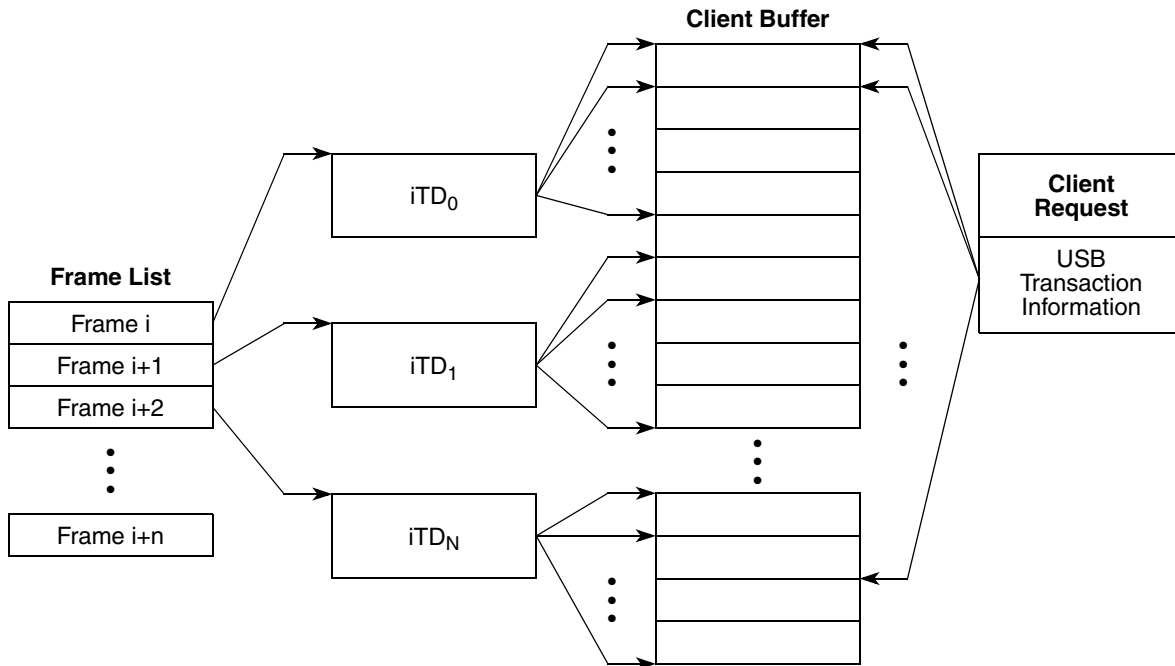


Figure 39-58. Example Association of iTDs to Client Request Buffer

As noted earlier, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the transaction offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction, it selects a transaction description record based on FRINDEX[2:0]. It then uses the current page buffer pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available page buffer pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer wraps a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

39.6.6.2.1 Periodic Scheduling Threshold

The isochronous scheduling threshold field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. System software uses it when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and continue to have the host controller process them.

The iTD and siTD data structures each describe eight micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span eight micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the isochronous scheduling threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame), but always dumps any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as two micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the isochronous scheduling threshold field. In the frame-caching model, system software assumes the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N , if the current micro-frame is 0 to 6, software can safely add isochronous transactions to Frame $N + 1$. If the current micro-frame is 7, software can add isochronous transactions to Frame $N + 2$.

Micro-frame caching is indicated with a non-zero value in the least-significant three bits of the isochronous scheduling threshold field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the isochronous scheduling threshold field. For example, if the count value were two, then the host controller keeps a window of two micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

39.6.7 Asynchronous Schedule

The asynchronous schedule traversal is enabled or disabled via the asynchronous schedule enable bit in the USBCMD register. If the asynchronous schedule enable bit is cleared, the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if the asynchronous schedule enable bit is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to the asynchronous schedule enable bit are not necessarily immediate. The new value of the bit is only taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

The asynchronous schedule status bit in the USBSTS register indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to the asynchronous schedule enable bit in the USBCMD register. Software can then poll the asynchronous schedule status bit to determine when the asynchronous schedule has made the desired transition. Software must not modify the asynchronous schedule enable bit unless the value of the asynchronous schedule enable bit equals that of the asynchronous schedule status bit.

The asynchronous schedule manages all control and bulk transfers. Control and bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when the asynchronous schedule enable bit is cleared.

Software may only write this register with defined results when the schedule is disabled. For example, asynchronous schedule enable bit in the USBCMD and the asynchronous schedule status bit in the USBSTS register are cleared. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting the asynchronous schedule enable bit is set. The asynchronous schedule is actually enabled when the asynchronous schedule status bit is set.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition
- The schedule has been disabled via the Asynchronous Schedule Enable bit in the USBCMD register.

The queue heads in the asynchronous list are linked into a simple circular list as shown in [Figure 39-54](#). Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTID or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

39.6.7.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. System software writes the physical memory address of a queue head into the ASYNCLISTADDR register, then enables the list by setting the asynchronous schedule enable bit in the USBCMD register to a one.

When inserting a queue head into an active list, software must ensure the schedule is always coherent from the host controllers' point of view. This means the system software must ensure all queue head pointer fields are valid. For example, qTD pointers have T-Bits set or reference valid qTDs and the horizontal pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)
--
-- Requirement: all inputs must be properly initialized.
--
-- pQHeadCurrent is a pointer to a queue head that is
-- already in the active list
-- pQHeadNew is a pointer to the queue head to be added
--
-- This algorithm links a new queue head into a existing
-- list
--
pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer
pQueueHeadCurrent.HorizontalPointer = physicalAddressOf (pQueueHeadNew)
End InsertQueueHead
```

39.6.7.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. Software deactivates the asynchronous schedule by setting the asynchronous schedule enable bit in the USBCMD register to a zero. Software can determine when the list is idle when the asynchronous schedule status bit in the USBSTS register is cleared. The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list using the following algorithm. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```
UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)
--
-- Requirement: all inputs must be properly initialized.
--
-- pQHeadPrevious is a pointer to a queue head that
-- references the queue head to remove
-- pQHeadToUnlink is a pointer to the queue head to be
-- removed
-- pQHeadNext is a pointer to a queue head in the
-- schedule. Software provides this pointer with the
-- following strict rules:
-- if the host software is one queue head, then
-- pQHeadNext must be the same as
-- QueueheadToUnlink.HorizontalPointer. If the host
```



```

-- software is unlinking a consecutive series of
-- queue heads, QHeadNext must be set by software to
-- the queue head remaining in the schedule.
--
-- This algorithm unlinks a queue head from a circular list
--
pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
End UnlinkQueueHead

```

If software removes the queue head with the H-bit set, it must select another queue head linked into the schedule and set its H-bit. This should be completed before removing the queue head. The requirement is that software keep one queue head in the asynchronous schedule, with its H-bit set. At the point software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

The handshake is implemented with three bits in the host controller. The first bit is a command bit (interrupt on async advance doorbell bit in the USBCMD register) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (interrupt on async advance bit in the USBSTS register) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures recently removed. When the host controller sets this status bit, it also clears the command bit. The third bit is an interrupt enable (interrupt on async advance bit in the USBINTR register) that is matched with the status bit. If the status bit is set and the interrupt enable bit is set, the host controller asserts a hardware interrupt.

[Figure 39-59](#) illustrates a general example where consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that remains in the asynchronous schedule.

When the host controller observes that doorbell bit being set, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (traversed beyond queue head (B) in this example).

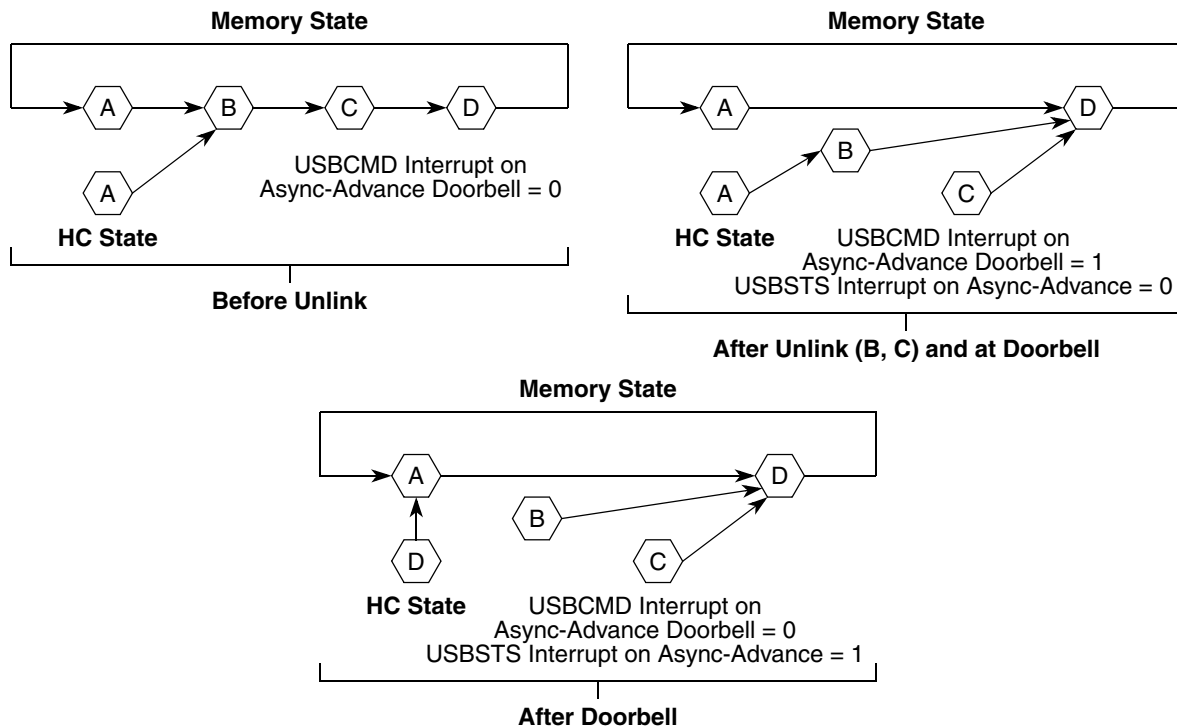


Figure 39-59. Generic Queue Head Unlink Scenario

Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue twice) before setting the advance on async status bit.

Software may re-use the memory associated with the removed queue heads after it observes the interrupt on async advance status bit is set, following assertion of the doorbell. Software should acknowledge the interrupt on async advance status as indicated in the USBSTS register, before using the doorbell handshake again.

39.6.7.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see [Figure 39-51](#)) defines an H-bit in the queue head that allows software to mark a queue head as being the head of the reclaim list. Host controller also keeps a 1-bit flag in the USBSTS register (reclamation) that is cleared when the host controller observes a queue head with the H-bit set. The reclamation flag in the status register is set when any USB transaction from the asynchronous schedule is executed or when the asynchronous schedule starts, see [Section 39.6.7.5, “Asynchronous Schedule Traversal: Start Event.”](#)

If the controller ever encounters an H-bit of one and a reclamation bit of zero, the controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is shown in [Figure 39-60](#).

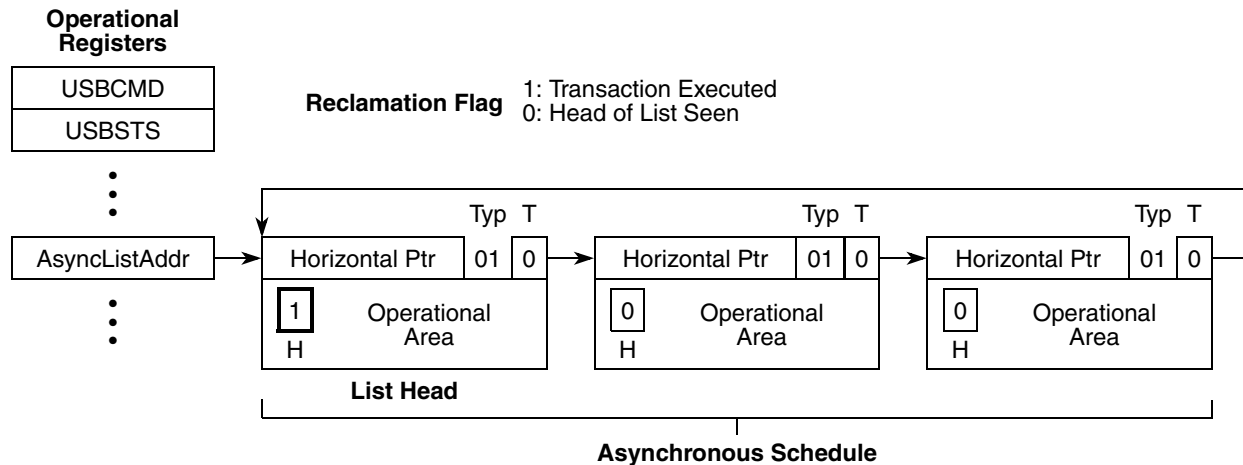


Figure 39-60. Asynchronous Schedule List with Annotation to Mark Head of List

39.6.7.4 Restarting Asynchronous Schedule Before EOF

There are many situations where the host controller detects an empty list before the end of the micro-frame. It is important to remember that under many circumstances the schedule traversal has stopped due to Nak/Nyet responses from all endpoints.

An example of particular interest is when a start-split for a bulk endpoint occurs early in the micro-frame. Given the EHCI simple traversal rules, the complete-split for that transaction may Nak/Nyet out quickly. If it is the only item in the schedule, then the host controller ceases traversal of the Asynchronous schedule early in the micro-frame. To provide reasonable service to this endpoint, the host controller should issue the complete-split before the end of the current micro-frame, instead of waiting until the next micro-frame. When the reason for host controller idling asynchronous schedule traversal is because of empty list detection, it is mandatory the host controller implement a 'waking' method to resume traversal of the asynchronous schedule. An example method is described below.

39.6.7.4.1 Example Method for Restarting Asynchronous Schedule Traversal

The reason for idling the host controller when the list is empty is to keep the host controller from unnecessarily occupying too much memory bandwidth. How long should the host controller stay idle before restarting?

The answer in this example is based on deriving a manifest constant, which is the amount of time the host controller remains idle before restarting traversal. In this example, the manifest constant is called AsyncSchedSleepTime, and has a value of 10msec. The value is derived based on the analysis in [Section 39.6.7.4.2, “Example Derivation for AsyncSchedSleepTime”](#). The traversal algorithm is simple:

- Traverse the Asynchronous schedule until the either an End-Of-micro-Frame event occurs, or an empty list is detected. If the event is an End-of-micro-Frame, go attempt to traverse the Periodic schedule. If the event is an empty list, then set a sleep timer and go to a schedule sleep state.
- When the sleep timer expires, set working context to the Asynchronous Schedule start condition and go to schedule active state. The start context allows the HC to reload Nakent fields, etc. so the HC has a chance to run for more than one iteration through the schedule.

This process simply repeats itself each micro-frame. [Figure 39-61](#) illustrates a sample state machine to manage the active and sleep states of the Asynchronous Schedule traversal policy. There are three states: Actively traversing the Asynchronous schedule, Sleeping, and Not Active. The last two are similar in terms of interaction with the Asynchronous schedule, but the Not Active state means that the host controller is busy with the Periodic schedule or the Asynchronous schedule is not enabled. The Sleeping state is a special state where the host controller is waiting for a period of time before resuming execution of the Asynchronous schedule.

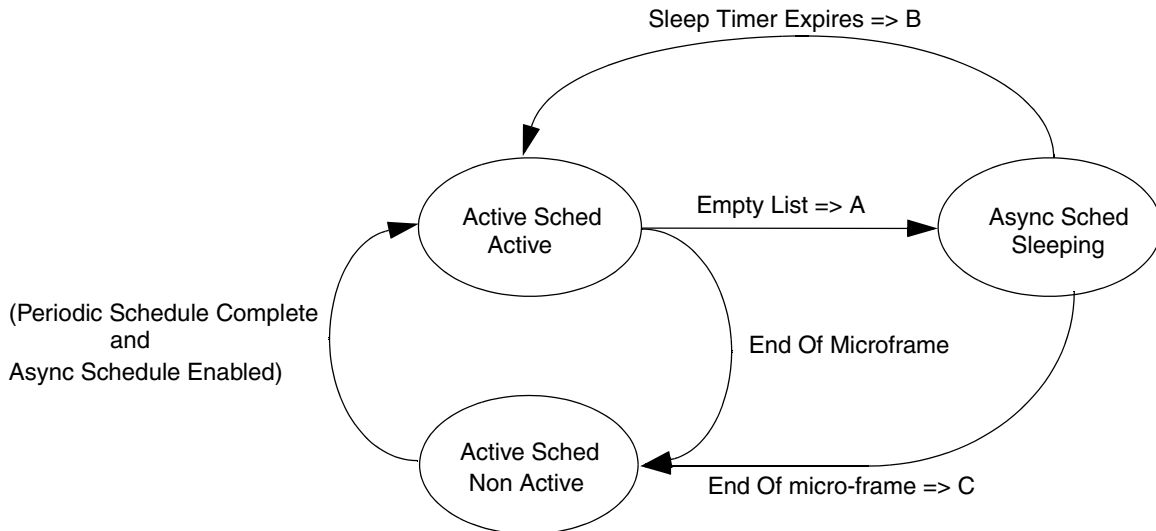


Figure 39-61. Example State Machine for Managing Asynchronous Schedule Traversal

Table 39-73. Asynchronous Schedule State Machine Transition Actions

Action	Action Description Label
A	On detection of the empty list, the host controller sets the <i>AsynchronousTraversalSleepTimer</i> to <i>AsyncSchedSleepTime</i> .
B	When the <i>AsynchronousTraversalSleepTimer</i> expires, the host controller sets the <i>Reclamation</i> bit in the USBSTS register to a one and moves the Nak Counter reload state machine to WaitForListHead (see 39.6.8, “Operational Model for NAK Counter).
C	The host controller cancels the sleep timer (<i>AsynchronousTraversalSleepTimer</i>).

Async Sched Not Active

This is the initial state of the traversal state machine after a host controller reset. The traversal state machine does not leave this state when the Asynchronous Schedule Enable bit in the USBCMD register is a zero.

This state is entered from Async Sched Active or Async Sched Sleeping states when the end-of-micro-frame event is detected.

Async Sched Active

This state is entered from the Async Sched Not Active state when the periodic schedule is not active. It is also entered from the Async Sched Sleeping states when the *AsynchronousTraversalSleepTimer* expires.

On every transition into this state, the host controller sets the Reclamation bit in the USBSTS register to a one.

While in this state, the host controller continually traverses the asynchronous schedule until either the end of micro-frame or an empty list condition is detected.

Async Sched Sleeping

The state is entered from the Async Sched Active state when a schedule empty condition is detected. On entry to this state, the host controller sets the AsynchronousTraversalSleepTimer to AsyncSchedSleepTime.

39.6.7.4.2 Example Derivation for AsyncSchedSleepTime

The derivation is based on analysis of what work the host controller could be doing next. It assumes the host controller does not keep any state about what work is possibly pending in the asynchronous schedule. The schedule could contain any mix of the possible combinations of high- full- or low-speed control and bulk requests. Table 39-74 summarizes some of the typical 'next transactions' that could be in the schedule, and the amount of time (e.g. footprint, or wall clock) the transaction takes to complete.

Table 39-74. Typical Low-/Full Speed Transaction Times

Transaction Attributes	Footprint (time)	Description
Speed : HS Size : 512	11.9 us	Maximum foot print for a worst case, full sized bulk data.
Speed : HS Size : 512	9.45 us	Maximum foot print for a best case, full sized bulk data.
Speed : FS Size : 64 Type : Bulk	Approx. 50 us	Approximate, typical for full sized bulk data.
Speed : FS Size : 8 Type : Cntrl	Approx. 12 us	Approximate, typical for 8 byte bulk/control (i.e. setup)

An AsyncSchedSleepTime value of 10 ms provides a reasonable relaxation of the system memory load and provides a good level of service for the various transfer types and payload sizes. For example, say you detect an empty list after issuing a start-split for a 64-byte full-speed bulk request. Assuming this is the only thing in the list, the host controller receives the results of the full-speed transaction from the hub during the fifth complete-split request. If the full-speed transaction was an IN and it nak'd, the 10ms sleep period would allow the host controller to get the NAK results on the first complete-split.

39.6.7.5 Asynchronous Schedule Traversal: Start Event

After the host controller has idled itself using the empty schedule detection, it naturally activates and begins processing from the periodic schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame), the host

controller must occasionally reactivate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule start events are defined to be:

- When the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

39.6.7.6 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature depends on the proper management of the Reclamation bit in the USBSTS register. The host controller tests for an empty schedule after it fetches a new queue head while traversing the asynchronous schedule. The host controller sets the Reclamation bit when an asynchronous schedule traversal start event occurs. The reclamation bit is also set when the host controller executes a transaction while traversing the asynchronous schedule. The host controller clears the reclamation bit when it finds a queue head with its H-bit set. Software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If software sets the H-bit in an interrupt queue head, the resulting behavior is undefined. The host controller may clear the reclamation bit when executing from the periodic schedule.

39.6.8 Operational Model for NAK Counter

This section describes the operational model for the NakCnt field defined in a queue head (see Section Queue Head). Software should not use this feature for interrupt queue heads. This rule is not required to be enforced by the host controller.

USB protocol has built-in flow control via the Nak response by a device. There are several scenarios, beyond the Ping feature, where an endpoint may naturally Nak or Nyet the majority of the time. An example is the host controller management of the split transaction protocol for control and bulk endpoints. All bulk endpoints (High- or Full-speed) are serviced via the same asynchronous schedule. The time between the Start-split transaction and the first Complete-split transaction could be short (i.e. like when the endpoint is the only one in the asynchronous schedule). The hub NYETs (effectively Naks) the Complete-split transaction until the classic transaction is complete. This could result in the host controller thrashing memory, repeatedly fetching the queue head and executing the transaction to the Hub, which does not complete until after the transaction on the classic bus

completes.

There are two component fields in a queue head to support the throttling feature: a counter field (NakCnt), and a counter reload field (RL). NakCnt is used by the host controller as one of the criteria to determine whether or not to execute a transaction to the endpoint. There are two operational modes associated with this counter:

- Not Used. This mode is set when the RL field is zero. The host controller ignores the NakCnt field for any execution of transactions through a queue head with an RL field of zero. Software must use this selection for interrupt endpoints.
- Nak Throttle Mode. This mode is selected when the RL field is non-zero. In this mode, the value in the NakCnt field represents the maximum number of Nak or Nyet responses the host controller

tolerates on each endpoint. In this mode, the HC decrements the NakCnt field based on the token/handshake criteria listed in Table 34. The host controller must reload NakCnt when the endpoint successfully moves data (e.g. policy to reward device for moving data).

Table 39-75. NAKCnt Field Adjustment Rules

Token	Handshake NAK	Handshake NYET
IN/PING	Decrement NAKCnt	N/A, Protocol Error
OUT	Decrement NAKCnt	No Action Start
Split	Decrement NAKCnt	N/A, Protocol Error
Complete Split	No Action	Decrement NAKCnt

In summary, system software enables the counter by setting the reload field (RL) to a non-zero value. The host controller may execute a transaction if NakCnt is non-zero. The host controller does not execute a transaction if NakCnt is zero. The reload mechanism is described in detail in Section Nak Count Reload Control.

When all queue heads in the Asynchronous Schedule either exhausts all transfers or all NakCnt's go to zero, then the host controller detects an empty Asynchronous Schedule and idle schedule traversal (see Section Empty Asynchronous Schedule Detection).

Any time the host controller begins a new traversal of the Asynchronous Schedule, a Start Event is assumed, see Section Asynchronous Schedule Traversal : Start Event. Every time a Start-Event occurs, the Nak Count reload procedure is enabled.

39.6.8.1 Nak Count Reload Control

When the host controller reaches the Execute Transaction state for a queue head (meaning that it has an active operational state), it checks to determine whether the NakCnt field should be reloaded from RL (see Section Execute Transaction). If the answer is yes, then RL is copied into NakCnt. After the reload or if the reload is not active, the host controller evaluates whether to execute the transaction.

The host controller must reload nak counters in queue heads (Figure 39-50) during the first pass through the reclamation list after an asynchronous schedule Start Event (see Section 39.6.7.5, “Asynchronous Schedule Traversal: Start Event” for the definition of the Start Event). The Asynchronous Schedule should have at most one queue head marked as the head (Figure 39-54). Figure 39-62 illustrates an example state machine that satisfies the operational requirements of the host controller detecting the first pass through the Asynchronous Schedule. This state machine is maintained internal to the host controller and is only used to gate reloading of the nak counter during the queue head traversal state: Execute Transaction (Figure 39-63). The host controller does not perform the nak counter reload operation if the RL field (see Figure 39-50) is set to zero.

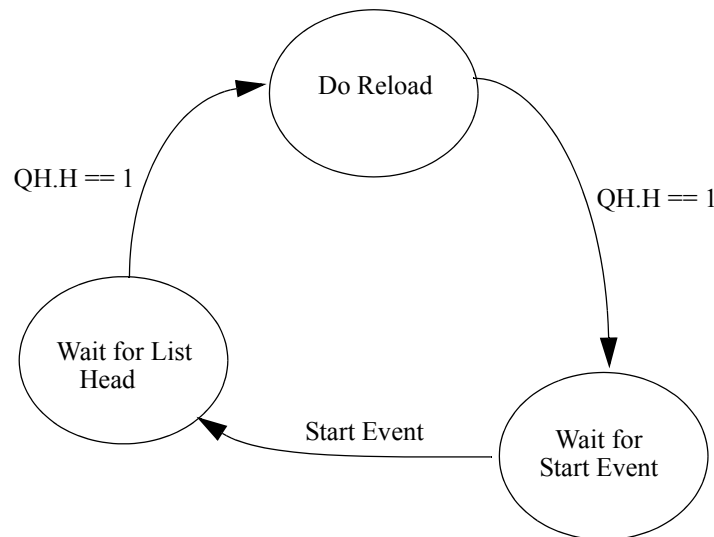


Figure 39-62. Example HC State Machine for Controlling Nak Counter Reloads

39.6.8.1.1 Wait for List Head

This is the initial state. The state machine enters this state from Wait for Start Event when a start event as defined in Section Asynchronous Schedule Traversal : Start Event occurs. The purpose of this state is to wait for the first observation of the head of the Asynchronous Schedule. This occurs when the host controller fetches a queue head whose H-bit is set to a one.

39.6.8.1.2 Do Reload

This state is entered from the Wait for List Head state when the host controller fetches a queue head with the H-bit set to a one. While in this state, the host controller performs nak counter reloads for every queue head visited that has a non-zero nak reload value (RL) field.

39.6.8.1.3 Wait for Start Event

This state is entered from the Do Reload state when a queue head with the H-bit set to a one is fetched. While in this state, the host controller does not perform nak counter reloads.

39.6.9 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the queue element transfer descriptor (qTD) structure defined in [Section 39.5.5, “Queue Element Transfer Descriptor \(qTD\).”](#)

One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed. Each qTD represents one or more bus transactions defined in the context of the EHCI specification as a transfer.

The general processing model for the host controller's use of a queue head is simple:

- Read a queue head
- Execute a transaction from the overlay area
- Write back the results of the transaction to the overlay area
- Move to the next queue head

If the host controller encounters errors during a transaction, the host controller sets one of the error reporting bits in the queue head's status field. The status field accumulates all errors encountered during the execution of a qTD (error bits in the queue head status field are sticky until the transfer [qTD] has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (for example, buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions occur for the endpoint and the host controller does not advance the queue.

An example host controller operational state machine of a queue head traversal is illustrated in [Figure 39-63](#). This state machine is a model for how a host controller should traverse a queue head. The host controller must be able to advance the queue from the Fetch QH state to avoid all hardware/software race conditions. This simple mechanism allows software to simply link qTDs to the queue head and activate them, then the host controller always finds them if/when they are reachable.

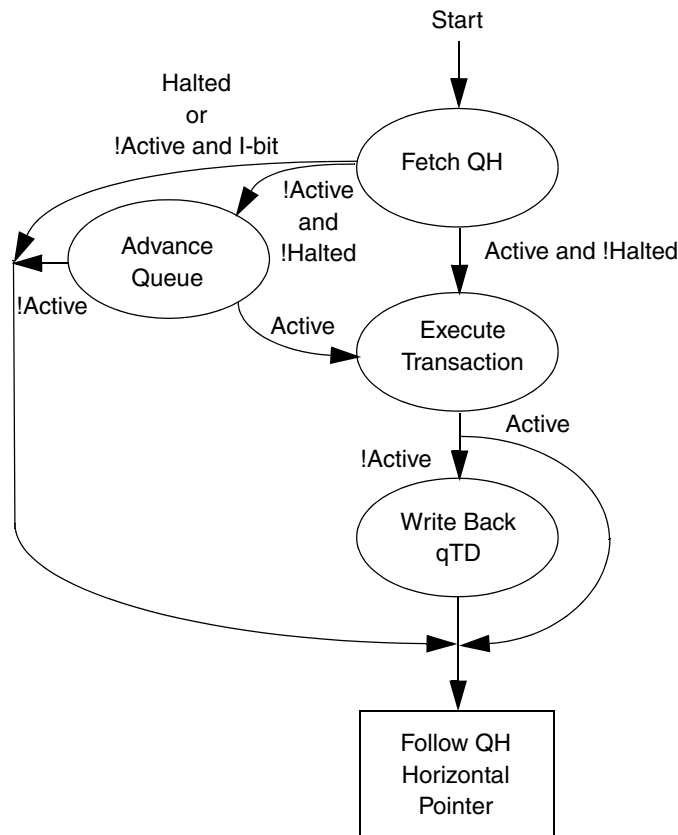


Figure 39-63. Host Controller Queue Head Traversal State Machine

This traversal state machine applies to all queue heads, regardless of transfer type or whether split transactions are required. The following sections describe each state. Each state description describes the entry criteria. The Execute Transaction state (Section Execute Transaction) describes the basic requirements for all endpoints. Sections Split Transactions for Asynchronous Transfers and Split Transaction Interrupt describe details of the required extensions to the Execute Transaction state for endpoints requiring split transactions.

NOTE

Prior to software placing a queue head into either the periodic or asynchronous list, software must ensure the queue head is properly initialized. Minimally, the queue head should be initialized to the following (see [Section 39.5.6, “Queue Head”](#) for layout of a queue head):

- Valid static endpoint state
- For the first use of a queue head, software may zero-out the queue head transfer overlay, then set the *Next qTD Pointer* field value to reference a valid qTD.

39.6.9.1 Fetch Queue Head

A queue head can be referenced from the physical address stored in the ASYNCLISTADDR Register (Section 3.6.6.2). Additionally, it may be referenced from the *Next Link Pointer field* of an iTD, siTD, FSTN or another Queue Head. If the referencing link pointer has the *Typ* field set to indicate a queue head, it is assumed to reference a queue head structure as defined in [Figure 39-51](#).

While in this state, the host controller performs operations to implement empty schedule detection (Section EmptyAsynchronous Schedule Detection) and Nak Counter reloads (Section Operational Model for Nak Counter). After the queue head has been fetched, the host controller conducts the following queries for empty schedule detection:

- If queue head is not an interrupt queue head (i.e. S-mask is a zero), and
- The H-bit is a one, and
- The Reclamation bit in the USBSTS register is a zero.

When these criteria are met, the host controller stops traversing the asynchronous list (as described in Section Empty Asynchronous Schedule Detection). When the criteria are not met, the host controller continues schedule traversal. If the queue head is not an interrupt and the H-bit is a one and the Reclamation bit is a one, then the host controller sets the Reclamation bit in the USBSTS register to a zero before completing this state. The operations for reloading of the Nak Counter are described in detail in Section Operational Model for Nak Counter.

This state is complete when the queue head has been read on-chip.

39.6.9.2 Advance Queue

To advance the queue, the host controller must find the next qTD, adjust pointers, perform the overlay and write back the results to the queue head.

This state is entered from the FetchQHD state if the overlay *Active* and *Halt* bits are set to zero. On entry to this state, the host controller determines which next pointer to use to fetch a qTD, fetches a qTD and determines whether or not to perform an overlay. If the I-bit is a one and the Active bit is a zero, the host controller immediately skips processing of this queue head, exits this state and uses the horizontal pointer to the next schedule data structure. If the field Bytes to Transfer is not zero and the T-bit in the Alternate Next qTD Pointer is set to zero, then the host controller uses the Alternate Next qTD Pointer. Otherwise, the host controller uses the Next qTD Pointer. If Next qTD Pointer's T-bit is set to a one, then the host controller exits this state and uses the horizontal pointer to the next schedule data structure.

Using the selected pointer the host controller fetches the referenced qTD. If the fetched qTD has its Active bit set to a one, the host controller moves the pointer value used to reach the qTD (Next or Alternate Next) to the Current qTD Pointer field, then performs the overlay. If the fetched qTD has its Active bit set to a zero, the host controller aborts the queue advance and follows the queue head's horizontal pointer to the next schedule data structure. The host controller performs the overlay based on the following rules:

- The value of the data toggle (dt) field in the overlay area depends on the value of the data toggle control (dtc) bit.
- If the EPS field indicates the endpoint is a high-speed endpoint, the Ping state field is preserved by the host controller. The value of this field is not changed as a result of the overlay.
- C-prog-mask field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- Frame Tag field is set to zero (field from incoming qTD is ignored, as is the current contents of the overlay area).
- NakCnt field in the overlay area is loaded from the RL field in the queue head's Static Endpoint State.
- All other areas of the overlay are set by the incoming qTD.

The host controller exits this state when it has committed the write to the queue head.

39.6.9.3 Execute Transaction

The host controller enters this state from the Fetch Queue Head state only if the *Active* bit in Status field of the queue head is set to a one.

On entry to this state, the host controller executes a few pre-operations, then checks some pre-condition criteria before committing to executing a transaction for the queue head.

The pre-operations performed and the pre-condition criteria depend on whether the queue head is an interrupt endpoint. The host controller can determine that a queue head is an interrupt queue head when the queue head's S-mask field contains a non-zero value. It is the responsibility of software to ensure the S-mask field is appropriately initialized based on the transfer type. There are other criteria that must be met if the EPS field indicates that the endpoint is a low- or full-speed endpoint, see [Section 39.6.11.1, "Split Transactions for Asynchronous Transfers"](#) and [Section 39.6.11.2, "Split Transaction Interrupt"](#).

- Interrupt Transfer Pre-condition Criteria

If the queue head is for an interrupt endpoint (e.g. non-zero *S-mask* field), then the FRINDEX[2:0] field must identify a bit in the S-mask field that has a one in it. For example, an S-mask value of

00100000b would evaluate to true only when FRINDEX[2:0] is equal to 101b. If this condition is met then the host controller considers this queue head for a transaction.

- Asynchronous Transfer Pre-operations and Pre-condition Criteria

If the queue head is not for an interrupt endpoint (e.g. a zero S-mask field), then the host controller performs one pre-operation and then evaluates one pre-condition criteria: The pre-operation is:

- Checks the Nak counter reload state. It may be necessary for the host controller to reload the Nak Counter field. The reload is performed at this time.

The pre-condition evaluated is:

- Whether or not the NakCnt field has been reloaded, the host controller checks the value of the NakCnt field in the queue head. If NakCnt is non-zero, or if the Reload Nak Counter field is zero, then the host controller considers this queue head for a transaction.

- Transfer Type Independent Pre-operations

Regardless of the transfer type, the host controller always performs at least one pre-operation and evaluates one pre-condition. The pre-operation is:

- A host controller internal transaction (down) counter qHTransactionCounter is loaded from the queue head's Mult field. A host controller implementation is allowed to ignore this for queue heads on the asynchronous list. It is mandatory for interrupt queue heads. Software should ensure that the Mult field is set appropriately for the transfer type.

The pre-conditions evaluated are:

- The host controller determines whether there is enough time in the micro-frame to complete this transaction. If there is not enough time to complete the transaction, the host controller exits this state.
- If the value of qHTransactionCounter for an interrupt endpoint is zero, then the host controller exits this state.

When the pre-operations are complete and pre-conditions are met, the host controller sets the Reclamation bit in the USBSTS register to a one and then begins executing one or more transactions using the endpoint information in the queue head. The host controller iterates qHTransactionCounter times in this state executing transactions. After each transaction is executed, qHTransactionCounter is decremented by one. The host controller exits this state when one of the following events occurs:

- The qHTransactionCounter decrements to zero, or
- The endpoint responds to the transaction with any handshake other than an ACK*, or
- The transaction experiences a transaction error, or
- The Active bit in the queue head goes to a zero, or
- There is not enough time in the micro-frame left to execute the next transaction (see Section Transaction Fit - A Best-Fit Approximation Algorithm for example method for implementing the frame boundary test).

NOTE

For a high-bandwidth interrupt OUT endpoint, the host controller may optionally immediately retry the transaction if it fails.

The results of each transaction is recorded in the on-chip overlay area. If data was successfully moved during the transaction, the transfer state in the overlay area is advanced. To advance queue head's transfer state, the Total Bytes to Transfer field is decremented by the number of bytes moved in the transaction, the data toggle bit (dt) is toggled, the current page offset is advanced to the next appropriate value (e.g. advanced by the number of bytes successfully moved), and the C_Page field is updated to the appropriate value (if necessary). See [Section 39.6.9.6, “Buffer Pointer List Use for Data Streaming with qTDs”](#).

The total bytes to transfer field may be zero when all the other criteria for executing a transaction are met. When this occurs, the host controller executes a zero-length transaction to the endpoint. If the PID_Code field indicates an IN transaction and the device delivers data, the host controller detects a packet babble condition, set the babble and halted bits in the Status field, set the Active bit to a zero, write back the results to the source qTD, then exit this state.

In the event an IN token receives a data PID mismatch response, the host controller must ignore the received data (e.g. not advance the transfer state for the bytes received). Additionally, if the endpoint is an interrupt IN, then the host controller must record that the transaction occurred (e.g. decrement qHTransactionCounter). It is

recommended (but not required) the host controller continue executing transactions for this endpoint if the resultant value of *qHTransactionCounter* is greater than one.

If the response to the IN bus transaction is a Nak (or Nyet) and RL is non-zero, NakCnt is decremented by one. If RL is zero, then no write-back by the host controller is required (for a transaction receiving a Nak or Nyet response and the value of CErr did not change). Software should set the RL field to zero if the queue head is an interrupt endpoint. Host controller hardware is not required to enforce this rule or operation.

After the transaction has finished and the host controller has completed the post processing of the results (advancing the transfer state and possibly NakCnt, the host controller writes back the results of the transaction to the queue head's overlay area in main memory.

The number of bytes moved during an IN transaction depends on how much data the device endpoint delivers. The maximum number of bytes a device can send is Maximum Packet Size. The number of bytes moved during an OUT transaction is either Maximum Packet Length bytes or Total Bytes to Transfer, whichever is less.

If there was a transaction error during the transaction, the transfer state (as defined above) is not advanced by the host controller. The CErr field is decremented by one and the status field is updated to reflect the type of error observed. Transaction errors are summarized in [Section Transaction Error](#).

The following events causes the host controller to clear the Active bit in the queue head's overlay status field. When the Active bit transitions from a one to a zero, the transfer in the overlay is considered complete. The reason for the transfer completion (clearing the Active bit) determines the next state.

- CErr field decrements to zero. When this occurs the Halted bit is set to a one and Active is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.
- The device responds to the transaction with a STALL PID. When this occurs, the Halted bit is set to a one and the Active bit is set to a zero. This results in the hardware not advancing the queue and the pipe halts. Software must intercede to recover.

- The Total Bytes to Transfer field is zero after the transaction completes. For a zero length transaction, it was zero before the transaction was started. When this condition occurs, the Active bit is set to zero.
- The PID code is an IN, and the number of bytes moved during the transaction is less than the Maximum Packet Length. When this occurs, the Active bit is set to zero and a short packet condition exists. The short-packet condition is detected during the Advance Queue state. Refer to Section 5.12 for additional rules for managing low- and full-speed transactions.
- The PID Code field indicates an IN and the device sends more than the expected number of bytes (e.g. Maximum Packet Length or Total Bytes to Transfer bytes, whichever is less) (e.g. a packet babble). This results in the host controller setting the Halted bit to a one.

With the exception of a NAK response (when RL field is zero), the host controller always writes the results of the transaction back to the overlay area in main memory. This includes when the transfer completes. For a high-speed endpoint, the queue head information written back includes minimally the following fields:

- NakCnt, dt, Total Bytes to Transfer, C_Page, Status, CERR, and Current Offset

For a low- or full-speed device the queue head information written back also includes the fields:

- C-prog-mask, FrameTag and S-bytes.

The duration of this state depends on the time it takes to complete the transaction(s) and the status write to the overlay is committed.

39.6.9.3.1 Halting a Queue Head

A halted endpoint is defined only for the transfer types that are managed via queue heads (control, bulk and interrupt). The following events indicate that the endpoint has reached a condition where no more activity can occur without intervention from the driver:

- An endpoint may return a STALL handshake during a transaction,
- A transaction had three consecutive error conditions, or
- A Packet Babble error occurs on the endpoint.

When any of these events occur (for a queue head) the Host Controller halts the queue head and set the USBERRINT status bit in the USBSTS register to a one. To halt the queue head, the Active bit is set to a zero and the Halted bit is set to a one. There may be other error status bits that are set when a queue is halted. The host controller always writes back the overlay area to the source qTD when the transfer is complete, regardless of the reason (normal completion, short packet or halt). The host controller does not advance the transfer state on a transaction that results in a *Halt* condition (e.g. no updates necessary for Total Bytes to Transfer, C_Page, Current Offset, and dt). The host controller must update CErr as appropriate. When a queue head is halted, the USB Error Interrupt bit in the USBSTS register is set to a one. If the USB Error Interrupt Enable bit in the USBINTR register is set to a one, a hardware interrupt is generated at the next interrupt threshold.

39.6.9.3.2 Asynchronous Schedule Park Mode

Asynchronous Schedule Park mode is a special execution mode that can be enabled by system software, where the host controller is permitted to execute more than one bus transaction from a high-speed queue

head in the Asynchronous schedule before continuing horizontal traversal of the Asynchronous schedule. This feature has no effect on queue heads or other data structures in the Periodic schedule. This feature is similar in intent as the Mult feature that is used in the Periodic schedule. Where-as the Mult feature is a characteristic that is tunable for each endpoint; park-mode is a policy that is applied to all high-speed queue heads in the asynchronous schedule. It is essentially the specification of an iterator for consecutive bus transactions to the same endpoint. All of the rules for managing bus transactions and the results of those as defined in Section Execute Transaction apply. This feature merely specifies how many consecutive times the host controller is permitted to execute from the same queue head before moving to the next queue head in the Asynchronous List. This feature should allow the host controller to attain better bus utilization for those devices that are capable of moving data at maximum rate, while at the same time providing a fair service to all endpoints.

A host controller exports its capability to support this feature to system software by setting the Asynchronous Schedule Park Capability bit in the HCCPARAMs register to a one. This information keys system software that the Asynchronous Schedule Park Mode Enable and Asynchronous Schedule Park Mode Count fields in the USBCMD register are modifiable. System software enables the feature by writing a one to the Asynchronous Schedule Park Mode Enable bit.

When park-mode is not enabled (e.g. Asynchronous Schedule Park Mode Enable bit in the USBCMD register is a zero), the host controller must not execute more than one bus transaction per high-speed queue head, per traversal of the asynchronous schedule. When park-mode is enabled, the host controller must not apply the feature to a queue head whose EPS field indicates a Low/Full-speed device (i.e. only one bus transaction is allowed from each Low/Full-speed queue head per traversal of the asynchronous schedule). Park-mode may only be applied to queue heads in the Asynchronous schedule whose EPS field indicates that it is a high-speed device.

The host controller must apply park mode to queue heads whose EPS field indicates a high-speed endpoint. The maximum number of consecutive bus transactions a host controller may execute on a high-speed queue head is determined by the value in the Asynchronous Schedule Park Mode Count field in the USBCMD register. Software must not set Asynchronous Schedule Park Mode Enable bit to a one and also set Asynchronous Schedule Park Mode Count field to a zero. The resulting behavior is not defined. An example behavioral example describes the operational requirements for the host controller implementing park-mode. This feature does not affect how the host controller handles the bus transaction as defined in Section Execute Transaction. It only effects how many consecutive bus transactions for the current queue head can be executed. All boundary conditions, error detection and reporting applies as usual. This feature is similar in concept to the use of the Mult field for high-bandwidth Interrupt for queue heads in the Periodic Schedule.

The host controller effectively loads an internal down-counter PM-Count from Asynchronous Schedule Park Mode Count when Asynchronous Schedule Park Mode Enable bit is a one, and a high-speed queue head is first fetched and meets all the criteria for executing a bus transaction. After the bus transaction, PM-Count is decremented. The host controller may continue to execute bus transactions from the current queue head until PM-Count goes to zero, an error is detected, the buffer for the current transfer is exhausted or the endpoint responds with a flow-control or STALL handshake.

39.6.9.4 Write Back qTD

This state is entered from the Execute Transaction state when the *Active* bit is set to a zero. The source data for the write-back is the transfer results area of the queue head overlay area (Section Figure 39-51, “Queue Head Layout”). The host controller uses the Current qTD Pointer field as the target address for the qTD. The queue head transfer result area is written back to the transfer result area of the target qTD. This state is also referred to as: qTD retirement. The fields that must be written back to the source qTD include Total Bytes to Transfer, Cerr, and Status.

The duration of this state depends on when the qTD write-back has been committed.

39.6.9.5 Follow Queue Head Horizontal Pointer

The host controller must use the horizontal pointer in the queue head to the next schedule data structure when any of the following conditions exist:

- If the Active bit is a one on exit from the Execute Transaction state, or
- When the host controller exits the Write Back qTD state, or
- If the Advance Queue state fails to advance the queue because the target qTD is not active, or
- If the Halted bit is a one on exit from the Fetch QH state.

There is no functional requirement that the host controller wait until the current transaction is complete before using the horizontal pointer to read the next linked data structure. However, it must wait until the current transaction is complete before executing the next data structure.

39.6.9.6 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers used to reference the data buffer for a transfer. The EHCI specification requires the buffer associated with the transfer be virtually contiguous. If the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

Figure 39-64 illustrates these requirements.

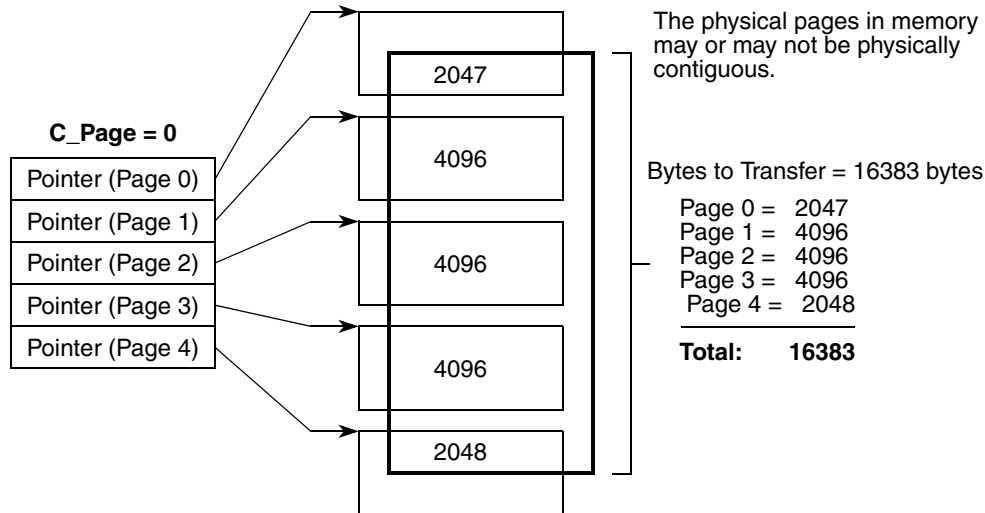


Figure 39-64. Example Mapping of qTD Buffer Pointers to Buffer Pages

The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD manages a 16 Kbyte buffer with any starting buffer alignment.

The host controller uses the C_Page field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is true even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically moves to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the bytes to transfer field.

Figure 39-64 illustrates a nominal example of how system software would initialize the buffer pointers list and the C_Page field for a transfer size of 16383 bytes. C_Page is cleared. The upper 20-bits of Page 0 references the start of the physical page. Current offset (the lower 12-bits of queue head doubleword 7) holds the offset in the page for example, 2049 (for example, 4096-2047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is cleared) and concatenates the current offset field. The 512 bytes are moved during the transaction, and the current offset and total bytes to transfer are adjusted by 512 and written back to the queue head working area.

During the fourth transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller increments C_Page (to 1) and uses the page 1 pointer to move the final byte of the transaction. After the fourth transaction, the active page pointer is the page 1 pointer and current offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller manages C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

The only valid adjustment the host controller may make to C_Page is to increment by one.

39.6.9.7 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's S-Mask to indicate which micro-frame within a one millisecond period a transaction should be executed for the queue head. Software must ensure all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in [Table 39-76](#).

Table 39-76. Example Periodic Reference Patterns for Interrupt Transfers

Frame Reference Sequence	Description
0, 2, 4, 6, 8,... S-Mask = 0x01	A queue head for the bInterval of two milliseconds (16 micro-frames) is linked into the periodic schedule so it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8,... S-Mask = 0x02	Another example of a queue head with a bInterval of two milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the S-Mask is set to 0x02 indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

39.6.9.8 Managing Transfer Complete Interrupts from Queue Heads

The host controller sets an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an interrupt on complete (IOC) bit set, or when a transfer (qTD) completes with a short packet. If system software needs multiple qTDs to complete a client request (like a control transfer) the intermediate qTDs do not require interrupts. System software may only need a single interrupt to notify it that the complete buffer has been transferred. System software may set IOCs to occur more frequently. A motivation for this may be that it wants early notification so interface data structures can be re-used in a timely manner.

39.6.10 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called ping. Ping is required for all USB 2.0 high-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The status field has a ping state bit that the host controller uses to determine the next actual PID it uses in the next transaction to the endpoint (see [Table 39-61](#)). The ping state bit is only managed by the host controller for queue heads that meet all of the following criteria:

- The queue head is not an interrupt
- The EPS field equals High-Speed
- The PIDCode field equals OUT

[Table 39-77](#) illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the *USB Specification, Revision 2.0* for detailed description on the Ping protocol.

Table 39-77. Ping Control State Transition Table

Current	Event		Next
	Host	Device	
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr ¹	Do Ping
Do Ping	PING	Stall	N/C ²
Do OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping ³
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr ¹	Do Ping
Do OUT	OUT	Stall	N/C ²

¹ Transaction Error (XactErr) is any time the host misses the handshake.

² No transition change required for the ping state bit. The stall handshake results in the endpoint being halted (for example, active cleared and halt set). Software intervention is required to restart queue.

³ A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and transition the ping state bit to do ping.

The ping state bit is described in [Table 39-61](#). The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (start in Do OUT when you don't know whether there is space on the device or not). The host controller manages the ping state bit. System software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the ping state bit across all queue advancements. This means when a new qTD is written into the queue head overlay area, the previous value of the ping state bit is preserved.

39.6.11 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB 1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below a USB 2.0 hub, utilizing the split transaction protocol. Refer to the USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and low-speed devices are enumerated identically as high-speed devices, but the transactions to the full- and low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the full- or low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and transaction translator below that the full- or low-speed device is attached.

EHCI uses dedicated data structures for managing full-speed isochronous data streams. Control, bulk, and interrupt are managed using the queuing data structures. The interface data structures need to be programmed with the device address and the transaction translator number of the USB 2.0 hub operating as the low-/full-speed host controller for this link. The following sections describe how the host controller processes and manages the split transaction protocol.

39.6.11.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an EPS field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full- and low-speed control are managed via queue heads in the asynchronous schedule.

Software must initialize the queue head with the appropriate device address and port number for the transaction translator serving as the full-/low-speed host controller for the links connecting the endpoint. Software must also initialize the split transaction state bit (SplitXState) to Do-Start-Split. Finally, if the endpoint is a control endpoint, system software must set the control transfer type (C) bit in the queue head to a one. If this is not a control transfer type endpoint, the C bit must be initialized by software to be a zero. The host controller uses this information to properly set the endpoint type (ET) field in the split transaction bus token. When the C bit is a zero, the split transaction token's ET field is set to indicate a bulk endpoint. When the C bit is a one, the split transaction token's ET field is set to indicate a control endpoint. Refer to Chapter 8 of *USB Specification, Revision 2.0* for details.



39.6.11.1.1 Asynchronous—Do-Start-Split

Do-Start-Split is the state that software must initialize a full- or low-speed asynchronous queue head. This state is entered from the Do-Complete-Split state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller executes a start-split transaction to the transaction translator. If the bus transaction completes without an error and PID code indicates an IN or OUT transaction, then the host controller reloads the error counter (Cerr). If it is a successful bus transaction and the PID Code indicates a SETUP, the host controller does not reload the error counter. If the transaction translator responds with a NAK, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response), the host controller decrements Cerr and proceeds to the next queue head in the asynchronous schedule.

39.6.11.1.2 Asynchronous - Do-Complete-Split

This state is entered from the Do-Start-Split state only after a start-split transaction receives an ACK handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset, and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PID code indicates an IN or OUT, the host controller reloads the error counter (Cerr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PID Code indicates a SETUP, the host controller must not adjust the value of Cerr.

Independent of PID code, the following responses have the indicated effects:

- Transaction Error (XactErr). Timeout/data CRC failure. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller ensures that the next time the host

controller begins executing from the asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule is the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.

- **NAK.** The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- **STALL.** The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte and retires the qTD, but does not attempt to advance the queue.

If the PID code indicates an IN, then any of following responses are expected:

- **DATA0/1.** On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced, and this state is exited.

If the PID code indicates an OUT/SETUP, any of following responses are expected:

- **ACK.** The target endpoint accepted the data, so the host controller must advance the state of the transfer. The current offset field is incremented by maximum packet length or bytes to transfer, whichever is less. The bytes to transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

39.6.11.2 Split Transaction Interrupt

Split-transaction interrupt-IN/OUT endpoints are managed using the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, the host controller visits a queue head, executes a high-speed transaction (if criteria are met), and advances the transfer state (or not) depending on the results of the entire transaction for a high-speed endpoint. For low- and full-speed endpoints, the details of the execution phase are different (that is, takes more than one bus transaction to complete), but the remainder of the operational framework is intact.

39.6.11.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the endpoint type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames or the data or response information in the pipeline is lost. Figure 39-66 illustrates the general scheduling boundary conditions supported by the EHCI periodic schedule and queue head data structure. The S and C_n labels indicate micro-frames where software can schedule start-splits and complete splits (respectively).

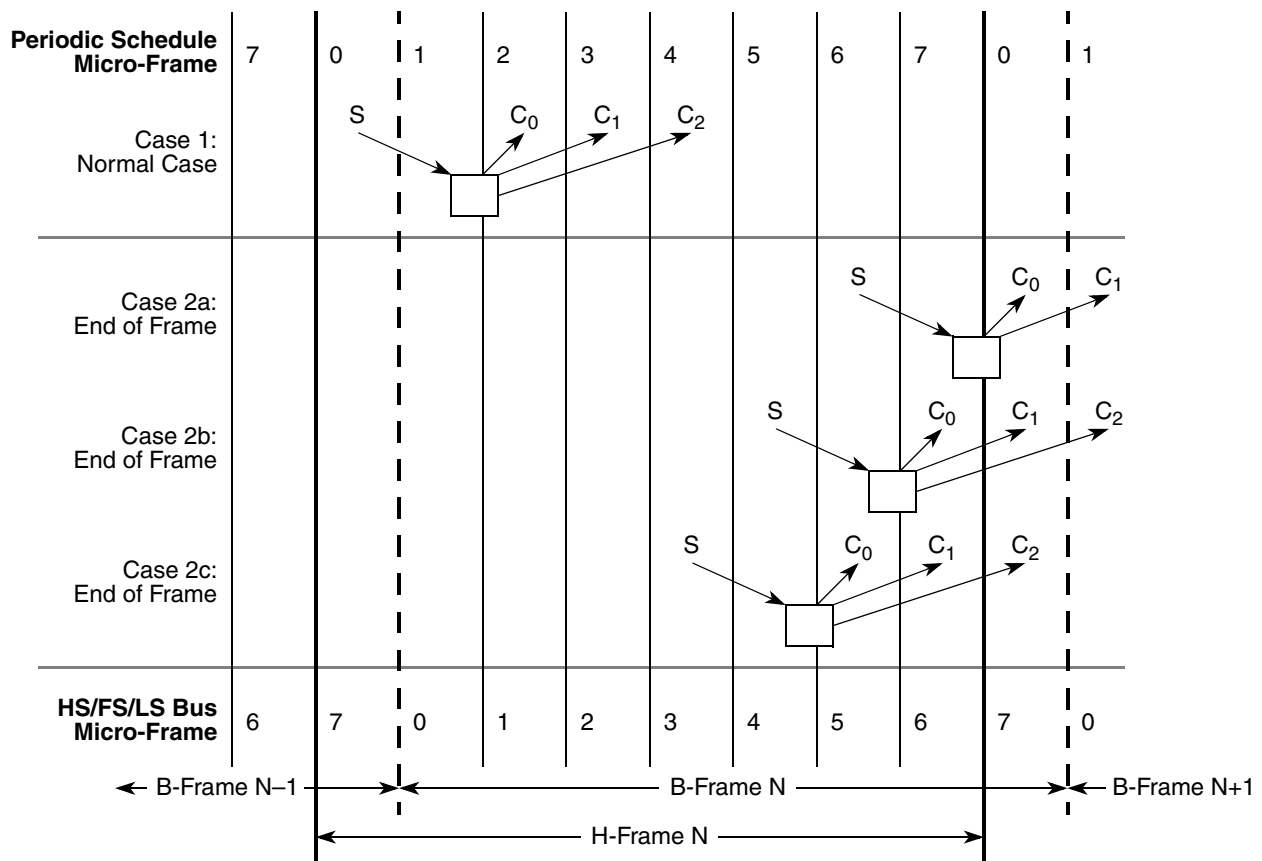


Figure 39-66. Split Transaction and Interrupt Scheduling Boundary Conditions

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).

- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient schedule that satisfies this requirement unless it uses FSTNs. Figure 39-67 illustrates the general layout of the periodic schedule.

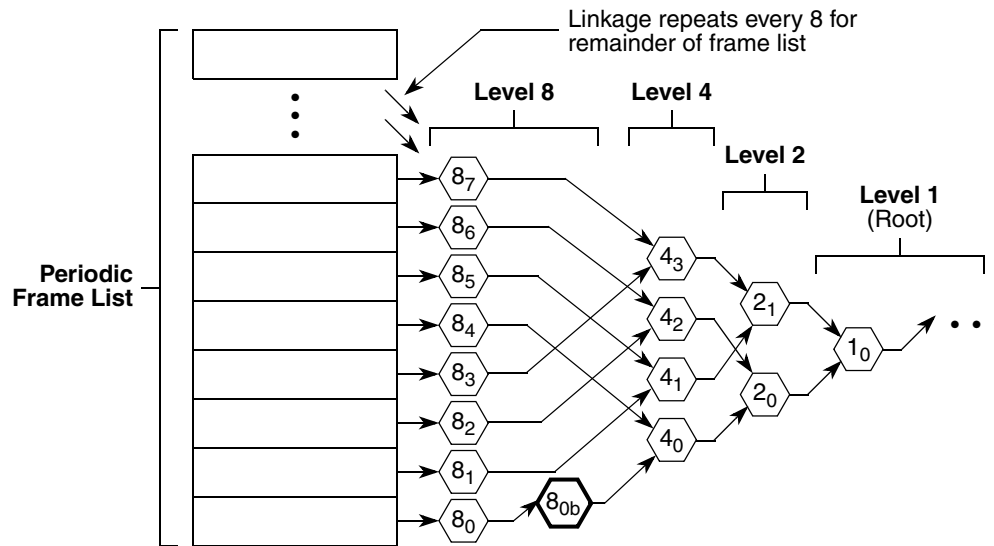


Figure 39-67. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading

The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a 2^N poll rate. Software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate eight queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 8_{0b} were such an endpoint. Without additional support on the interface, to get 8_{0b} reachable at the correct time, software would have to link 8_1 to 8_{0b} . It would then have to move 4_1 and everything linked after into the same path as 4_0 . This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. Section 39.5.7, “Periodic Frame Span Traversal Node (FSTN),” defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by system software to instruct the host controller when to execute portions of the split-transaction protocol.

- SplitXState. This is a single bit residing in the status field of a queue head (Table 39-61). This bit is used to track the current state of the split transaction.

- **Frame S-mask.** This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the status field of the queue head. For example, in case one of [Figure 39-66](#), the S-mask would have a value of 0b0000_0001, indicating that if the queue head is traversed by the host controller, the SplitXState indicates Do_Start, and the current micro-frame as indicated by FRINDEX[2:0] is 0 executes a start-split transaction.
- **Frame C-mask.** This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the status field of the queue head. For example, in case one of [Figure 39-66](#), the C-mask would have a value of 0b0001_1100, indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, or 4 executes a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

39.6.11.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure manages low/full-speed interrupt queue heads that need to be reached from consecutive frame list locations (boundary cases 2a through 2c). An FSTN is essentially a back pointer, similar in intent to the back pointer field in the siTD data structure.

This feature provides software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, and then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in [Section 39.5.7, “Periodic Frame Span Traversal Node \(FSTN\).”](#)
- A save place indicator; this is always an FSTN with its back path link pointer [T] bit cleared.
- A restore indicator; this is always an FSTN with its back path link pointer [T] bit set.
- Host controller FSTN traversal rules.

Host Controller Operational Model for FSTNs

When the host controller encounters an FSTN during micro-frames two through seven, it follows the node's normal path link pointer to access the next schedule data structure. The FSTN's normal path link pointer [T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a save-place FSTN in micro-frames 0 or 1, it saves the value of the normal path link pointer and sets an internal flag indicating it is executing in recovery path mode. Recovery path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in recovery path mode until it encounters a restore FSTN or it determines it has reached the end of the micro-frame.

The rules for schedule traversal and limited execution while in recovery path mode are:

- Always follow the normal path link pointer when it encounters an FSTN that is a save-place indicator. The host controller must not recursively follow save-place FSTNs. Therefore, while executing in recovery path mode, it must never follow an FSTN's back path link pointer.
- Do not process an siTD or iTD data structure; simply follow its next link pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; follow its horizontal link pointer.
- When a QH's EPS field indicates a full/low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (this applies whether the PID code indicates an IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. The host controller must not execute a start-split transaction while executing in recovery path mode. Refer to the *EHCI Specification* for special handling when in recovery path mode.
- Stop traversing the recovery path when it encounters an FSTN that is a restore indicator. The host controller unconditionally uses the saved value of the save-place FSTN's normal path link pointer when returning to the normal path traversal. The host controller must clear the context of executing a recovery path when it restores schedule traversal to the save-place FSTN's normal path link pointer.

If the host controller determines there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path and clears the context of executing a recovery path. The result is the host controller starts traversal at the frame list at the start of the next consecutive micro-frame.

An example traversal of a periodic schedule that includes FSTNs is illustrated in [Figure 39-68](#).

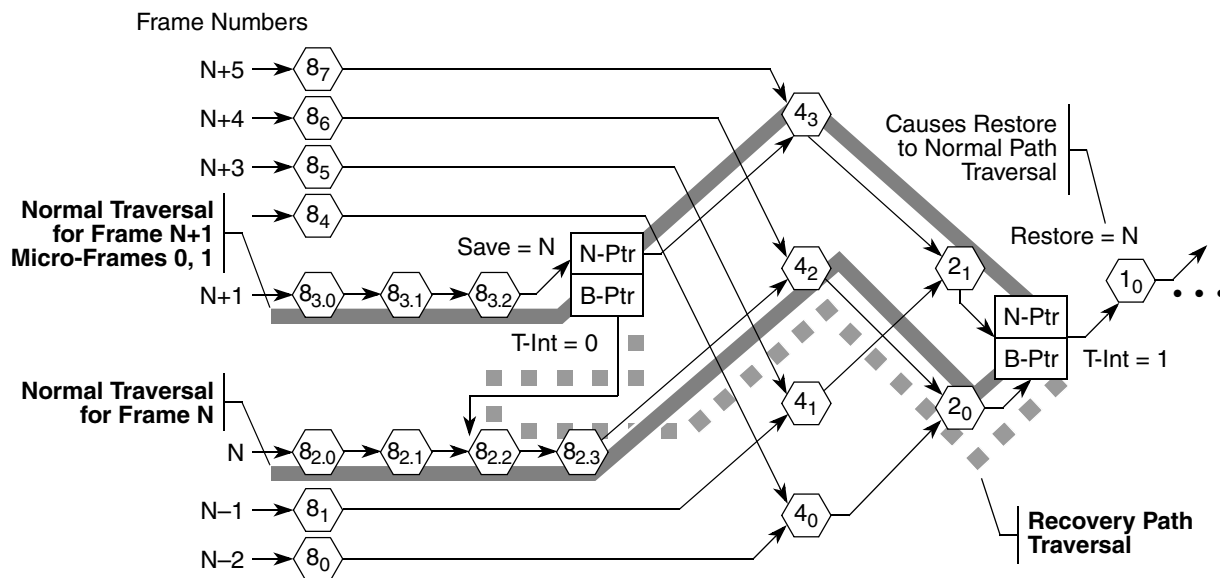


Figure 39-68. Example Host Controller Traversal of Recovery Path via FSTNs

In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the normal path link pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a save-place FSTN so it is not executing in recovery path mode. When

it encounters the restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N. Normal path link pointer to traverse to the next data structure (normal schedule traversal). This is because the host controller must use a restore FSTN's normal path link pointer when not executing in a recovery-path mode. The nodes traversed during frame N include: {8_{2,0}, 8_{2,1}, 8_{2,2}, 8_{2,3}, 4₂, 2₀, Restore-N, 1₀...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters save-path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N. Normal path link pointer and follows Save-N.Back Path Link Pointer. At the same time, it sets an internal flag indicating that it is now in recovery path mode (the recovery path is annotated in [Figure 39-68](#) with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits recovery path mode by clearing the internal recovery path mode flag and commences (restores) schedule traversal using the saved value of the save-place FSTN's normal path link pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {8_{3,0}, 8_{3,1}, 8_{3,2}, Save-A, 8_{2,2}, 8_{2,3}, 4₂, 2₀, Restore-N, 4₃, 2₁, Restore-N, 1₀...}.

In frame N+1 (micro-frames 2-7), when the host controller encounters save-path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {8_{3,0}, 8_{3,1}, 8_{3,2}, Save-A, 4₃, 2₁, Restore-N, 1₀...}.

Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each save-place indicator requires a matching restore indicator.
The save-place indicator is an FSTN with a valid back path link pointer and T-bit equal to zero. The Back path link pointer [Typ] field must be set to indicate the referenced data structure is a queue head. The restore indicator is an FSTN with its back path link pointer [T] bit set.
A restore FSTN may be matched to one or more save-place FSTNs. For example, if the schedule includes a poll-rate 1 level, system software only needs to place a restore FSTN at the beginning of this list to match all possible save-place FSTNs.
- If the schedule does not have elements linked at a poll-rate level of one, and one or more save-place FSTNs are used, system software must ensure the restore FSTN's normal path link pointer's T-bit is set, as this marks the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that recovery path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A save-place FSTN's back path link pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the save-place FSTN is reachable from frame list offset N, then the FSTN's back path link pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is software should have no more than one save-place FSTN reachable in any single frame. Two (or more, depending on the implementation) could exist as full/low-speed footprints change with bandwidth adjustments. This could occur, for example, when a bandwidth rebalance causes system software to move the save-place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

39.6.11.2.3 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue is halted, thus signaling system software to recover from the error. A data-loss condition exists when a start-split is issued, accepted, and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition. Instead, it affects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- **C-prog-mask.** This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector the host controller sets as one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- **FrameTag.** The host controller uses this field during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- **S-bytes.** This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

39.6.11.2.4 Split Transaction Execution State Machine for Interrupt

In the following section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous full- and low-speed endpoints, a split-transaction state machine manages the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that also manages the split transaction. The internal calculated mechanism is:

- **cMicroFrameBit.** This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX register, which is $\text{cMicroFrameBit} = 1 \text{ shifted-left}(\text{FRINDEX}[2:0])$. The cMicroFrameBit has at most one bit asserted that always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit equals 0b0000_0001.

The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 39-69 illustrates how a complete interrupt-split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction that occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH[S-mask]. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there is more than one complete-split transaction scheduled by software for the Do_Complete state. This translates simply to the fact that there are multiple bits set in the QH[C-mask] field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, therefore, requiring system software to detect the condition and perform system-dependent recovery).

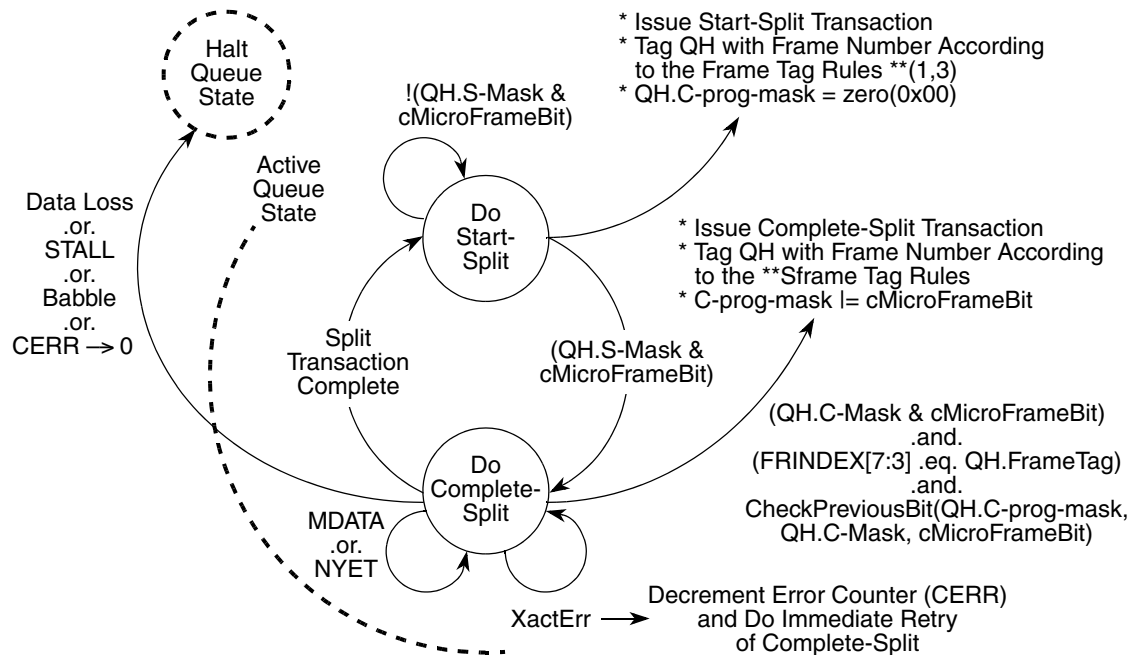


Figure 39-69. Split Transaction State Machine for Interrupt

Periodic Interrupt—Do-Start-Split

This is the state software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete. This occurs when one of the following events occur:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see [Section 39.6.11.2, “Split Transaction Interrupt.”](#)

Each time the host controller visits a queue head in this state (within the execute transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field, sets C-prog-mask to zero (0x00), and exits this state. The host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

Periodic Interrupt—Do-Complete-Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (within the execute transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH[C-mask] field. A non-zero result indicates software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH[FrameTag] is compared with the current contents of FRINDEX[7:3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```

Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask) then
    If not(previousBit bitAND QH.C-prog-mask) then
        rvalue = FALSE;
    End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
    rvalue = FALSE;
End if
return (rvalue)
End Algorithm
  
```

- Test D. Check if a start-split should be executed in this micro-frame. This is the same test performed in the Do Start Split state. When it evaluates to TRUE and the controller is NOT processing in the context of a recovery path mode, it means a start-split should occur in this micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If test A is non-zero, Test B indicates a match event, Test C indicates the previous complete-split was appropriately executed and Test D evaluates to a NOT TRUE condition, the host controller executes a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (any responses that result in decrementing of the Cerr result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.
- The test for whether this is the last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros, all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.
- NYET (and not Last). See above description for testing for last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.
- Transaction Error (XactErr). Timeout, data CRC failure, etc. The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero). If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and Cerr is not zero, this state is exited (return to Do Start Split). This results in a retry of the entire OUT split transaction at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The current offset field is incremented by maximum packet length or bytes to transfer, whichever is less. The bytes to transfer field is decremented by the same amount, and the data toggle bit (dt) is toggled. The host controller then exits this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.
- MDATA. This response occurs only for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.

- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
- Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
- If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced, and this state is exited.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits:

Active bit is cleared and the Halted bit is set and the qTD is retired. Responses not enumerated in the list or received out of sequence are illegal and may result in undefined host controller behavior. The other possible combinations of tests A, B, C, and D may indicate data or response was lost.

Table 39-78 lists the possible combinations and the appropriate action.

Table 39-78. Interrupt IN/OUT Do Complete Split State Execution Criteria

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current micro-frame. Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, the queue head must be halted. If PID code is an OUT, the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the missed micro-frame bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the missed micro-frame bit in the status field to a one.

Table 39-78. Interrupt IN/OUT Do Complete Split State Execution Criteria (continued)

Condition	Action	Description
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QH If PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PID Code is an IN, the queue head must be halted. If PID code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not affect Cerr. In either case, set the missed micro-frame bit in the status field to a one. When executing in the context of a recovery path mode, the host controller can process the queue head and take the actions indicated above or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a recovery path mode.

Managing the QH[FrameTag] Field

The QH[FrameTag] field in a queue head is completely managed by the host controller. The rules for setting QH[FrameTag] are simple:

- Rule 1: If transitioning from Do Start Split to Do Complete Split and the current value of FRINDEX[2:0] is 6, QH[FrameTag] is set to FRINDEX[7:3] + 1. This accommodates split transactions whose start-split and complete-splits are in different H-Frames (case 2a, see [Figure 39-66](#)).
- Rule 2: If the current value of FRINDEX[2:0] is 7, QH[FrameTag] is set to FRINDEX[7:3] + 1. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c in [Figure 39-66](#).
- Rule 3: If transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2:0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2:0]) is not 7, FrameTag is set to FRINDEX[7:3]. This accommodates all other cases in [Figure 39-66](#).

39.6.11.2.5 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (new S-mask and C-mask values).

It is imperative that system software not update these masks to new values in the midst of a split transaction. To avoid any race conditions with the update, the host controller provides a simple assist to system software. System software sets the inactivate-on-next-transaction (I) bit to signal the host controller it intends to update the S-mask and C-mask on this queue head. System software then waits for the host controller to observe the I-bit is set and transitions the active bit to a zero. The rules for how and when the host controller clears the active bit are:

- If the active bit is cleared, no action is taken. The host controller does not attempt to advance the queue when the I-bit is set.

- If the active bit is set and the SplitXState is DoStart (regardless of the value of S-mask), the host controller simply clears the active bit. The host controller is not required to write the transfer state back to the current qTD. If the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction; it must clear the active bit.

System software must save transfer state before setting the I-bit. This is required so it can correctly determine what transfer progress (if any) occurred after the I-bit was set, and the host controller executed its final bus-transaction and cleared the active bit.

After system software has updated the S-mask and C-mask, it must reactivate the queue head. Because the active bit and the I-bit cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the halted bit
2. Clear the I-bit
3. Set the active bit and clear the halted bit in the same write.

Setting the halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the active bit is set.

39.6.11.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The host controller uses siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (see [Section 39.6.6, “Managing Isochronous Transfers Using iTDs,”](#) for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

39.6.11.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in Section Split Transaction Scheduling Mechanisms for Interrupt apply. [Figure 39-70](#) illustrates the general scheduling boundary conditions supported by the EHCI periodic schedule. The S_n and C_n labels indicate micro-frames where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, and dashed line. The bottom of the figure illustrates the relationship of an siTD to the H-Frame.

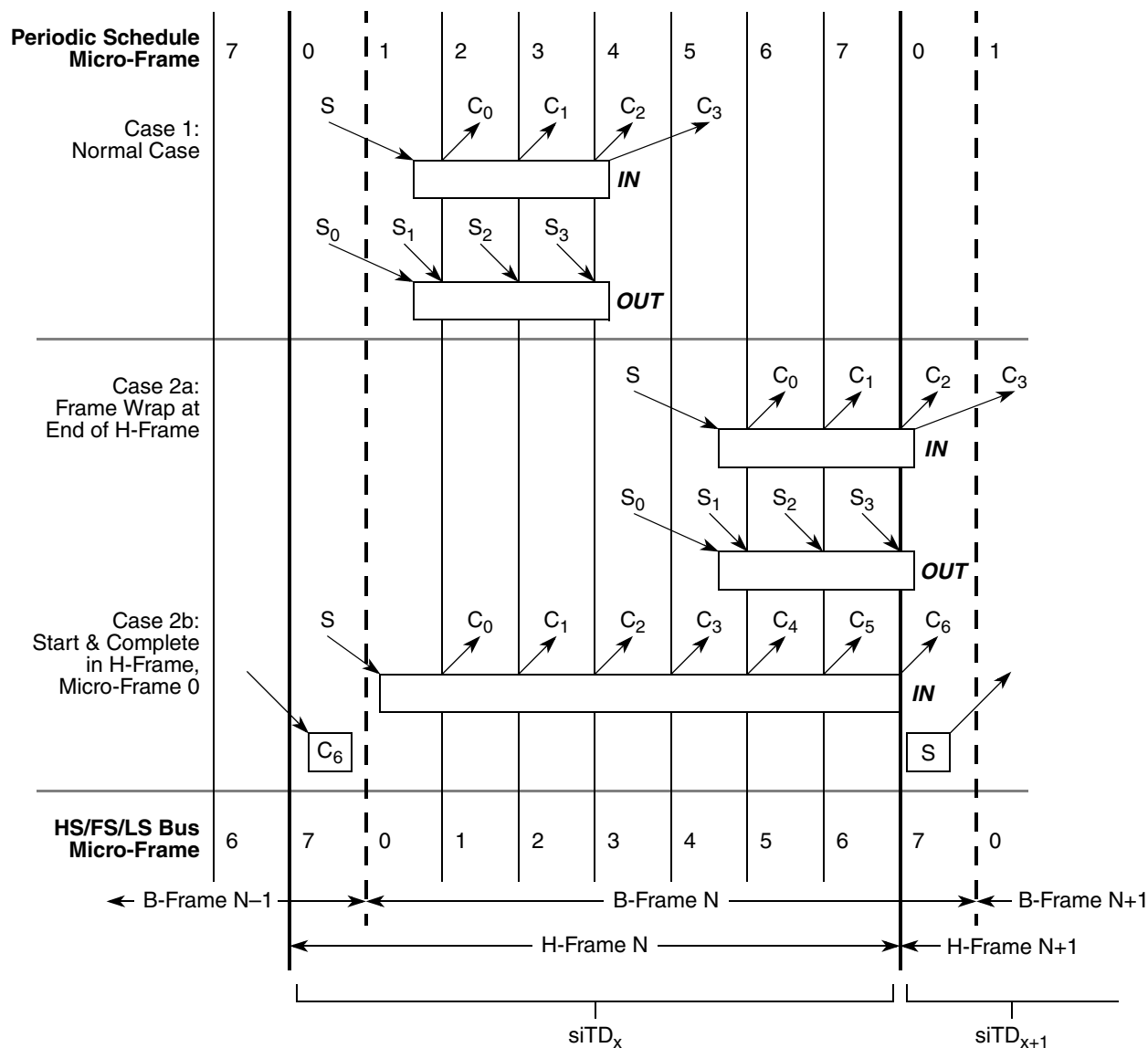


Figure 39-70. Split Transaction and Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list (for example, it takes two $siTD$ s in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.

- Case 2b: This case can only occur for a large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState. This is a single bit residing in the status field of an siTD (see [Table 39-55](#)). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in [Section 39.6.11.3.3, “Split Transaction Execution State Machine for Isochronous.”](#)
- Frame S-mask. This is a bit-field wherein system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in case one of [Figure 39-70](#), the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2:0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in case one of [Figure 39-70](#), the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2:0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN and OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. At the top of [Figure 39-71](#) are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. At the bottom is

illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

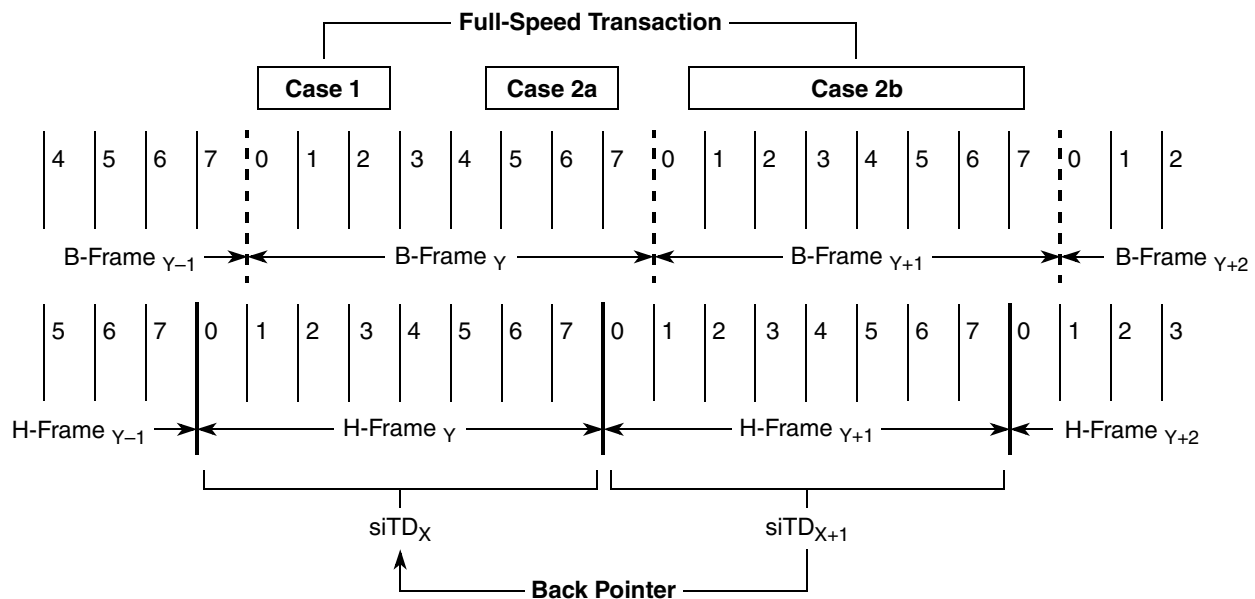


Figure 39-71. siTD Scheduling Boundary Examples

Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. The siTD_X is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-Frame_{Y+1}, or micro-frame 0 of H-Frame_{Y+2}. The complete splits are scheduled using siTD_{X+2} (not shown). The complete-splits to extract this data must use the buffer pointer from siTD_{X+1}. The only way for the host controller to reach siTD_{X+1} from H-Frame_{Y+2} is to use siTD_{X+2}'s back pointer.

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so it completes before the end of the B-Frame.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b could be scheduled so the start-split was in micro-frame 1 of H-Frame N and the last complete-split would need to occur in micro-frame 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and 2b, which has significant impact on the complexity of the host controller.

39.6.11.3.2 Tracking Split Transaction Progress for Isochronous Transfers

Isochronous endpoints do not employ the concept of a halt on error; however, the host controller does identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts, and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure that manages full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields transaction position (TP) and transaction count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there is more than one start-split transaction, each of which requires proper annotation. If host hold-offs occur, the sequence of annotations received from the host is not complete, which is detected and managed by the transaction translator. See [Section 39.6.11.3.1, “Split Transaction Scheduling Mechanisms for Isochronous,”](#) for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD[T-Count] and siTD[TP] are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. After the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split-transaction interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- **C-prog-mask.** This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2:0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so the remaining complete-splits are not executed. An IN siTD is retired based solely on the responses from the transaction translator to the complete-split transactions. This means, for example, it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD [total bytes to transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In another interface such as data structures (for example, high-speed data streams through queue heads), the transition of total bytes to transfer to zero signals the end of the transfer and results in clearing the active bit. However, in this case, the result has not been delivered by the transaction translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a transaction translator. In summary, the periodic pipeline rules require that on a micro-frame boundary, the transaction translator holds the final two bytes received (if it has not seen an end of packet [EOP]) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the transaction translator could have received the entire packet (including both CRC bytes), but not received the packet EOP. In the next micro-frame, the transaction translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes held in the full-speed pipeline stage). This could cause the siTD to decrement its total bytes to transfer field to zero, indicating it has received all expected data. The host must continue to execute one more (scheduled) complete-split transactions to extract the results of the full-speed transaction from the transaction translator (for example, the transaction translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller for it to report the hold-off event), system software must detect the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

39.6.11.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the active bit in the status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise, the host controller processes the siTD as specified below. A split transaction state machine manages the split-transaction protocol sequence. The host controller uses the fields defined in [Section 39.6.11.3.2, “Tracking Split Transaction Progress for Isochronous Transfers,”](#) plus the variable `cMicroFrameBit` defined in [Section 39.6.11.2.4, “Split Transaction Execution State Machine for Interrupt,”](#) to track the progress of an isochronous split transaction. [Figure 39-72](#) illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

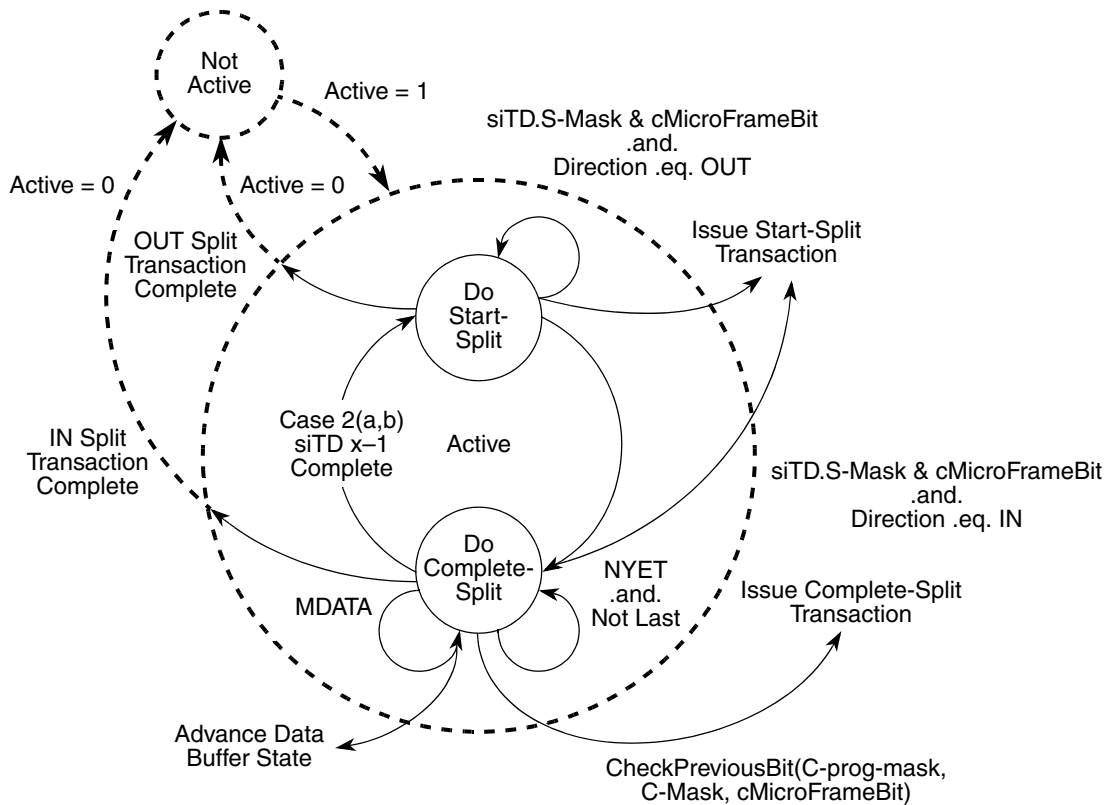


Figure 39-72. Split Transaction State Machine for Isochronous

Periodic Isochronous—Do-Start-Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is initialized to this state, the siTD transitions to this state from Do Complete Split when a case 2a (IN), or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the siTD [S-mask] against cMicroFrameBit. If there is a one in the appropriate position, the siTD executes a start-split transaction. By definition, the host controller cannot reach an siTD at the wrong time. If the I/O field indicates an IN, the start-split transaction includes only the extended token plus the full-speed token. Software must initialize the siTD [Total Bytes To Transfer] field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (the I/O field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating siTD [Current Offset] with the page pointer indicated by the page select field (siTD [P]). A zero in this field selects Page 0 and a one selects Page 1. During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the siTD [P] bit from a zero to a one, and begin using the siTD Page 1 with siTD [Current Offset] as the memory address pointer. The field siTD [TP] is used to annotate each start-split transaction with the indication of which part of the split-transaction data the current payload represents (ALL, BEGIN, MID,

END). In all cases, the host controller simply uses the value in siTD [TP] to mark the start-split with the correct transaction position code.

T-Count is always initialized to the number of start-splits for the current frame. TP is always initialized to the first required transaction position identifier. The scheduling boundary case (see [Figure 39-71](#)) determines the initial value of TP. The initial cases are summarized in [Table 39-79](#).

Table 39-79. Initial Conditions for OUT siTD's TP and T-count Fields

Case	T-count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates T-Count and TP appropriately so that the next start-split is correctly annotated. [Table 39-80](#) illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 39-80. Transaction Position (TP)/Transaction Count (T-Count) Transition Table

TP	T-count next	TP next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when T-count starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when T-count starts at greater than 2.
MID	!=1	MID	TP stays at MID while T-count is not equal to 1 (for example, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state, the following operations take place:

- The siTD [Total Bytes To Transfer] and the siTD [Current Offset] fields are adjusted to reflect the number of bytes transferred.
- The siTD [P] (page select) bit is updated appropriately.
- The siTD [TP] and siTD [T-count] fields are updated appropriately as defined in [Table 39-80](#).

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, total bytes to transfer). The host controller must clear the active bit

when it detects all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when total bytes to transfer decrements to zero. Either implementation must ensure that if the initial condition is total bytes to transfer equals zero and T-count equals one, the host controller issues a single start-split, with a zero-length data payload. Software must ensure that TP, T-count and total bytes to transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination yields undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer does not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in [Section 39.6.11.3.4, “Split Transaction for Isochronous - Processing Examples.”](#)

The host controller can optionally track the progress of an OUT split transaction by setting appropriate bits in the siTD [C-prog-mask] as it executes each scheduled start-split. The checkPreviousBit algorithm defined in [Section , “Periodic Isochronous - Do Complete Split,”](#) can be used prior to executing each start-split to determine if start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then clear the siTD's active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

Periodic Isochronous - Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied to depends on which micro-frame the host controller is currently executing, which means the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with the siTD [C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD in this state.
- Test B. The siTD [C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in [Section , “Periodic Interrupt—Do-Complete-Split”](#)). The sequence in which this test is applied depends on the current value of FRINDEX[2:0]. If FRINDEX[2:0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise, it is applied immediately.

```
Algorithm Boolean CheckPreviousBit(siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)
Begin
    Boolean rvalue = TRUE;
    previousBit = cMicroFrameBit rotate-right(1)
    -- Bit-wise anding previousBit with C-mask indicates whether there
    -- was an intent to send a complete split in the previous micro-
    -- frame. So, if the 'previous bit' is set in C-mask, check
    -- C-prog-mask to make sure it happened.
    if previousBit bitAND siTD.C-mask then
```

```

        if not (previousBit bitAND siTD.C-prog-mask) then
                                                    rvalue = FALSE
        End if
    End if
    Return rvalue
End Algorithm

```

If Test A is true and FRINDEX[2:0] is zero or one, this is a case 2a or 2b scheduling boundary (see Figure 39-70). See Section , “Complete-Split for Scheduling Boundary Cases 2a, 2b,” for details in managing this condition.

If Test A and Test B evaluate to true, the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD [Total Bytes To Transfer]
- Adjust siTD [Current Offset] by the number of bytes received
- Adjust the siTD [P] (page select) field if the transfer caused the host controller to use the next page pointer
- Set any appropriate bits in the siTD [Status] field, depending on the results of the transaction.

If the host controller encounters a condition where siTD [total bytes to transfer] is zero and it receives more data, the host controller must not write the additional data to memory. The siTD [Status-Active] bit must be cleared and the siTD [Status-Babble Detected] bit must be set. The fields siTD [Total Bytes To Transfer], siTD [Current Offset], and siTD [P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD [Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD [Status-Active] and set siTD [Status-Babble Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD [Status] field and clears the active bit.
- Transaction Error (XactErr). The complete-split transaction encounters a timeout, CRC16 failure, etc. The siTD [Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller does not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the active bit is cleared.
- DATAx (0 or 1). This response signals the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the active bit is cleared. If the bytes to transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), less data than was expected, or allowed for was actually received. This short packet event does not set

the USBINT status bit in the USBSTS register to a one. The host controller does not detect this condition.

- **NYET (and Last).** On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in section periodic interrupt - Do Complete Split. If it is the last complete-split (with a NYET response), the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- **MDATA (and Last).** See above description for testing for Last. This can only occur when there is an error condition. There has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction or software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the active bit.
- **NYET (and not Last).** See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- **MDATA (and not Last).** The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means one or more of the complete-splits have been skipped. The host controller sets the missed micro-frame status bit and clears the active bit.

Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see [Figure 39-70](#)) require the host controller use the transaction state context of the previous siTD to finish the split transaction. [Table 39-81](#) enumerates the transaction state fields.

Table 39-81. Summary siTD Split Transaction State

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) T-count (transaction count)	All bits in the status field	C-prog-mask

NOTE

TP and T-count are only for host to device (OUT) endpoints.

If software has budgeted the schedule of this data stream with a frame wrap case, it must initialize the siTD [Back Pointer] field to reference a valid siTD and have the T bit in the siTD [Back Pointer] field cleared. Otherwise, software must set the T bit in siTD [Back Pointer]. The host controller's rules for interpreting when to use the siTD [Back Pointer] field are listed below. These rules apply only when the siTD's active bit is a one and the SplitXState is Do Complete Split.

- When cMicroFrameBit is a 0x1 and the siTD_X[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTD_X[S-mask[0]] is zero

When either of these conditions apply, the host controller must use the transaction state from siTD_{X-1}.

To access siTD_{X-1}, the host controller reads on-chip the siTD referenced from siTD_X [Back Pointer].

The host controller must save the entire state from siTD_X while processing siTD_{X-1}. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD [Back Pointers].

If siTD_{X-1} is active (Active bit is set and SplitXStat is Do Complete Split), Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see [Table 39-81](#)) of siTD_{X-1} is appropriately advanced based on the results and written back to memory. If the resultant state of siTD_{X-1}'s active bit is a one, the host controller returns to the context of siTD_X and follows its next pointer to the next schedule item. No updates to siTD_X are necessary.

If siTD_{X-1} is active (Active bit is set and SplitXStat is Do Start Split), the host controller must clear the active bit and set the missed micro-frame status bit and the resultant status is written back to memory.

If siTD_{X-1}'s Active bit is cleared, (because it was cleared when the host controller first visited siTD_{X-1} via siTD_X's back pointer, it transitioned to zero as a result of a detected error, or the results of siTD_{X-1}'s complete-split transaction cleared it), the host controller returns to the context of siTD_X and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is 1 and siTD_X[S-mask[0]] is 1). If this criterion is met, the host controller immediately executes a start-split transaction and appropriately advances the transaction state of siTD_X, then follows siTD_X[Next Pointer] to the next schedule item. If the criterion is not met, the host controller simply follows siTD_X[Next Pointer] to the next schedule item. In the case of a 2b boundary case, the split-transaction of siTD_{X-1} has its active bit cleared when the host controller returns to the context of siTD_X. Software should not initialize an siTD with C-mask bits 0 and 1 set and an S-mask with bit 0 set. This scheduling combination is not supported and the behavior of the host controller is undefined.

39.6.11.3.4 Split Transaction for Isochronous - Processing Examples

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced using the execute transaction queue head traversal state machine.

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state

machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

The host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, [Table 39-82](#) illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 39-82. Example Case 2a - Software Scheduling siTDs for an IN Endpoint

siTDX		Micro-Frames								InitialSplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Because this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 0x10 (a one bit in micro-frame 4) and C-mask value of 0xC3 (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the back pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller visits the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. The siTD for frame X+1 has its SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it visits the second siTD eight times. During micro-frames 0 and 1, it detects that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that siTD_{X+1}'s back pointer [T] bit is a zero, saves the state of siTD_{X+1}, and fetches siTD_X. It executes the complete split transaction using the transaction state of siTD_X. If the siTD_X split transaction is complete, siTD's active bit is cleared and results written back to siTD_X. The host controller retains the fact that siTD_X is retired and transitions the SplitXState in siTD_{X+1} to Do Start Split. At this point, the host controller is prepared to execute the start-split for siTD_{X+1} when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in [Section , “Periodic Isochronous - Do Complete Split”](#)), before all the scheduled complete-splits have been executed, the host controller changes siTD_X[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split transactions. For this example, siTD_{X+1} does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame $X+2$, micro-frame 0, the host controller detects that $siTD_{X+2}$'s back pointer [T] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the active bit. The host controller returns to the context of $siTD_{X+2}$, and traverses its next pointer without any state change updates to $siTD_{X+2}$.

During H-Frame $X+2$, micro-frame 1, the host controller detects $siTD_{X+2}$'s S-mask[0] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and clears the Active bit. It returns to the state of $siTD_{X+2}$ and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for $siTD_{X+2}$ when it reaches micro-frame 4.

39.6.12 Host Controller Pause

When the host controller's HCHalted bit in the USBSTS register is a zero, the host controller is sending SOF (Start OF Frame) packets down all enabled ports. When the schedules are enabled, the EHCI host controller accesses the schedules in main memory each micro-frame. This constant ping-pong of main memory is known to create CPU power management problems for mobile systems. Specifically, mobile systems aggressively manage the state of the CPU, based on recent history usage. In the more aggressive power saving modes, the CPU can disable its caches. Current PC architectures assume that bus-master accesses to main memory must be cache-coherent. So, when bus masters are busy touching memory, the CPU power management software can detect this activity over time and inhibit the transition of the CPU into its lowest power savings mode. USB controllers are bus-masters and the frequency at which they access their memory-based schedules keeps the CPU power management software from placing the CPU into its lowest power savings state.

USB Host controllers don't access main memory when they are suspended. However, there are a variety of reasons why placing the USB controllers into suspend won't work, but they are beyond the scope of this document. The base requirement is that the USB controller needs to be kept out of main memory, while at the same time, the USB bus is kept from going into suspend.

EHCI controllers provide a large-grained mechanism that can be manipulated by system software to change the memory access pattern of the host controller. System software can manipulate the schedule enable bits in the USBCMD register to turn on/off the scheduling traversal. A software heuristic can be applied to implement an on/off duty cycle that allows the USB to make reasonable progress and allow the CPU power management to get the CPU into its lowest power state. This method is not intended to be applied at all times to throttle USB, but should only be applied in specific configurations and usage loads. For example, when only a keyboard or mouse is attached to the USB, the heuristic could detect times when the USB is attempting to move data only infrequently and can adjust the duty cycle to allow the CPU to reach its low power state for longer periods of time. Similarly, it could detect increases in the USB load and adjust the duty cycle appropriately, even to the point where the schedules are never disabled. The assumption here is that the USB is moving data and the CPU is required to process the data streams.

To provide a complete solution for the system, the companion host controllers should also provide a similar method to allow system software to inhibit the companion host controller from accessing its shared memory based data structures (schedule lists or otherwise).

39.6.13 Port Test Modes

EHCI host controllers implement the port test modes Test_J_State, Test_K_State, Test_Packet, Test_Force_Enable, and Test_SE0_NAK as described in the *USB Specification Revision 2.0*. The required, port test sequence is (assuming the CF-bit in the CONFIGFLAG register is set):

- Disable the periodic and asynchronous schedules by clearing the asynchronous schedule enable and periodic schedule enable bits in the USBCMD register.
- Place all enabled root ports into the suspended state by setting the suspend bit in each appropriate PORTSC register.
- Clear the run/stop bit in the USBCMD register and wait for the HCHalted bit in the USBSTS register, to transition to a one. An EHCI host controller implementation may optionally allow port testing with the run/stop bit set. However, all host controllers must support port testing with run/stop cleared and HCHalted set.
- Set the port test control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, the run/stop bit in the USBCMD register must be transitioned back to one to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (HCHalted bit is a one), it terminates and exits test mode by setting HCRreset.

39.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (Port change events, etc.), and
- Host controller error events

All transaction-based sources are maskable through the host controller's interrupt enable register (USBINTR). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter via the interrupt threshold control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer does not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller does not generate interrupts any more frequently than once every eight micro-frames.

[Section 39.6.14.2.4, “Host System Error”](#) details effects of a host system error.

If an interrupt is scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval has been written back to system memory. This may result in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion, it is assumed control is received. When the interrupt handler receives control, its first action is to read the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (via an OS-specific mechanism) schedules a deferred procedure call (DPC) that executes later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

NOTE

The only method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

39.6.14.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

39.6.14.1.1 Transaction Error

A transaction error is any error that caused the host controller to think the transfer did not complete successfully. [Table 39-83](#) lists the events/responses the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

Table 39-83. Summary of Transaction Errors

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	-1	XactErr set	1 ¹
Timeout	-1	XactErr set	1 ¹
Bad PID ²	-1	XactErr set	1 ¹
Babble	N/A	See Section , “Serial Bus Babble”	1
Buffer Error	N/A	See Section , “Data Buffer Error”	

¹ If occurs in a queue head, USBERRINT is asserted only when Cerr counts down from a one to a zero. In addition, the queue is halted.

² The host controller received a response from the device, but it could not recognize the PID as a valid PID.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the Cerr field is decremented. When the PID Code indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set and the Cerr field being decremented.

- EPS field indicates a high-speed device and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device and it returns a Nyet handshake to a SETUP.
- EPS field indicates a low- or full-speed device and the complete-split receives a Nak handshake.

Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a packet babble. When a device sends more data than the maximum length number of bytes, the host controller sets the babble detected bit to a one and halts the endpoint if it is using a queue head. Maximum length is defined as the minimum of total bytes to transfer and maximum packet size. The Cerr field is not decremented for a packet babble condition (only applies to queue heads). A babble condition also exists if IN transaction is in progress at high-speed EOF2 point. This is called a frame babble. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port to which the frame babble is detected.

The USBERRINT bit in the USBSTS register is set. If the USB error interrupt enable bit in the USBINTR register is set, a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

When a host controller detects a data PID mismatch, it must disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on maximum packet size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake to advance the transmitter's data sequence. The EHCI interface allows system software to provide buffers for a control, bulk, or interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. When a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. To properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

Data Buffer Error

This event indicates an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the data buffer error bit in the queue head, iTD or siTD.

If the data buffer error occurs on a non-isochronous IN, the host controller does not issue a handshake to the endpoint. This forces the endpoint to resend the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable. An acceptable implementation option is to one's complement the CRC bytes and send them.

There are other options suggested in the Transaction Translator section of the *USB Specification Revision 2.0*.

39.6.14.1.2 USB Interrupt (Interrupt on Completion (IOC))

Transfer descriptors (iTDs, siTDs, and queue heads (qTDs)) contain a bit that can cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes the USB Interrupt (USBINT) bit in the USBSTS register to be set. In addition, if a short packet is encountered on an IN transaction associated with a queue head, this event also causes USBINT to be set. If the USB interrupt enable bit in the USBINTR register is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, the USBERRINT bit in the USBSTS register is also set.

39.6.14.1.3 Short Packet

Reception of a data packet less than the endpoint's max packet size during control, bulk, or interrupt transfers signals the completion of the transfer. When a short packet completion occurs during a queue head execution, the USBINT bit in the USBSTS register is set. If the USB interrupt enable bit is set in the USBINTR register, a hardware interrupt is signaled to the system at the next interrupt threshold.

39.6.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold, with the one exception being the interrupt on async advance.

39.6.14.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set, the host controller sets the port change detect bit in the USBSTS register. If the port change interrupt enable bit in the USBINTR register is set, the host controller issues a hardware interrupt. The port status change bits are:

- Connect Status Change
- Port Enable/Disable Change
- Over-current Change
- Force Port Resume

39.6.14.2.2 Frame List Rollover

This event indicates the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. (If the frame list size is 1024, the interrupt occurs every 1024 milliseconds. If it is 512, it occurs every 512 milliseconds, etc.) When a frame list rollover is detected, the host controller sets the frame list rollover bit in the USBSTS register. If the frame list rollover enable bit in the USBINTR register is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

39.6.14.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. When the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of the interrupt on async advance doorbell bit in the USBCMD register. If it is set, it sets the interrupt on async advance bit in the USBSTS register. If the interrupt on async advance enable bit in the USBINTR register is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in [Section 39.6.7.2, “Removing Queue Heads from Asynchronous Schedule.”](#)

39.6.14.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error may be catastrophic to the host controller making it impossible for the host controller to continue in a coherent fashion. Behavior for these types of errors is to halt the host controller. Host-based error must result in the following actions:

- The run/stop bit in the USBCMD register is cleared.
- The host system error and HCHalted bits in the USBSTS register are set.
- If the host system error enable bit in the USBINTR register is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

[Table 39-84](#) summarizes the required actions taken on the various host errors.

Table 39-84. Summary Behavior on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal
siTD fetch (read)	Fatal	Fatal	Fatal
siTD status write-back (write)	Fatal	Fatal	Fatal
iTD fetch (read)	Fatal	Fatal	Fatal
iTD status write-back (write)	Fatal	Fatal	Fatal
qTD fetch (read)	Fatal	Fatal	Fatal
qHD status write-back (write)	Fatal	Fatal	Fatal
Data write	Fatal	Fatal	Fatal
Data read	Fatal	Fatal	Fatal

NOTE

After a host system error, software must reset the host controller using HCRreset in the USBCMD register before re-initializing and restarting the host controller.

39.7 Device Data Structures

This section defines the interface data structures that communicate control, status, and data between device controller driver (DCD) software and the device controller. Data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of device queue heads and transfer descriptors.

NOTE

Software must ensure no interface data structure reachable by the device controller spans a 4K-page boundary.

The data structures defined in this section are (from the device controller's perspective) a mix of read-only and read/writable fields. The device controller must preserve the read-only fields on all data structure writes.

The USB core includes DCD software called the USB 2.0 device API. The device API provides an easy to use application program interface for developing device (peripheral) applications. The device API incorporates and abstracts for the application developer all of the elements of the program interface.

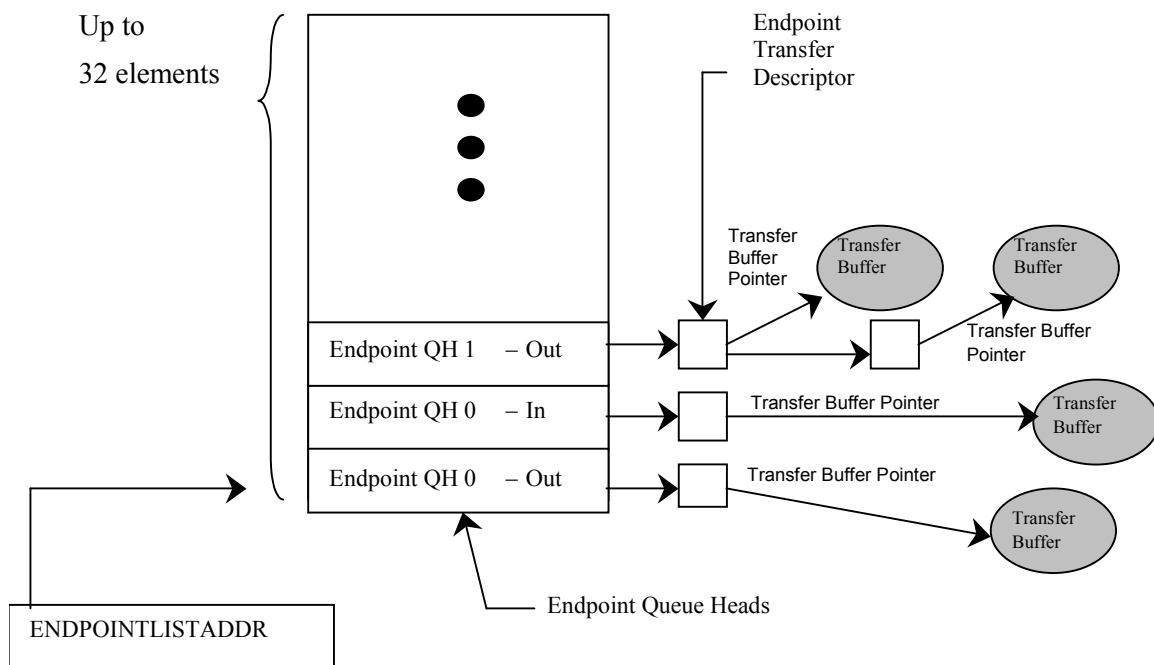


Figure 39-73. End Point Queue Head Organization

39.7.1 Endpoint Queue Head

The device Endpoint Queue Head (dQH) is where all transfers are managed. The dQH is a 48-byte data structure, but must be aligned on 64-byte boundaries. During priming of an endpoint, the dTD (device transfer descriptor) is copied into the overlay area of the dQH, which starts at the next TD pointer 32-bit word and continues through the end of the buffer pointers 32-bit words. After a transfer is complete, the

dTD status 32-bit word is updated in the dTD pointed to by the current TD pointer. While a packet is in progress, the overlay area of the dQH is a staging area for the dTD so the device controller can access needed information with little minimal latency.

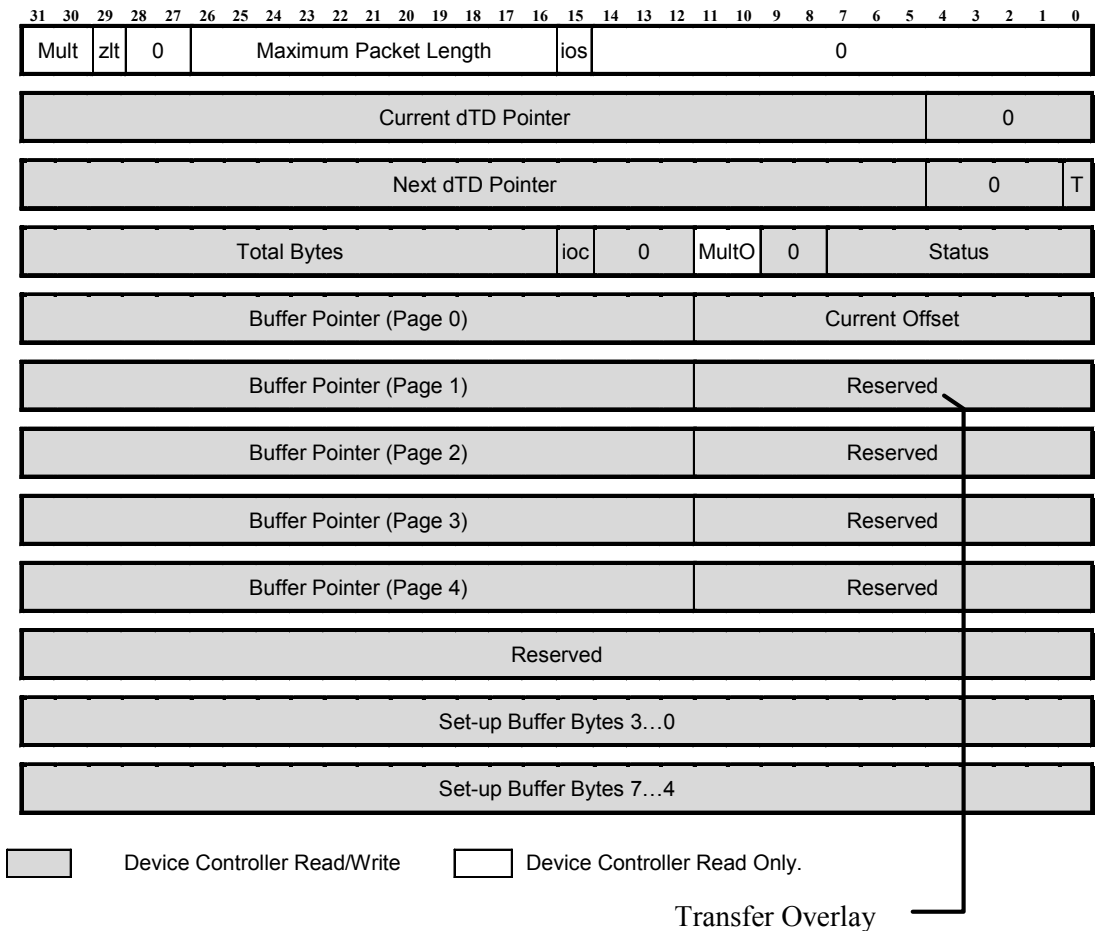


Figure 39-74. Endpoint Queue Head

39.7.1.1 Endpoint Capabilities/Characteristics

This 32-bit word specifies static information about the endpoint; in other words, this information does not change over the lifetime of the endpoint. Device controller software should not attempt to modify this information while the corresponding endpoint is enabled.

Table 39-85. Endpoint Capabilities/Characteristics

Bit	Description
31:30	Mult. This field indicates the number of packets executed per transaction description as given by the following: 00 Execute n transactions as demonstrated by the USB variable length packet protocol where n is computed using the maximum packet length (dQH) and the total bytes field (dTD) 01 Execute 1 Transaction 10 Execute 2 Transactions 11 Execute 3 Transactions Note: Non-ISO endpoints must set Mult equal to 00. Note: ISO endpoints must set Mult equal to 01, 10, or 11 as needed.
29	Zero Length Termination Select. This bit indicates when a zero length packet terminates transfers where total transfer length is a multiple. This bit is not relevant for isochronous transfers. 0 Enable zero length packet to terminate transfers equal to a multiple of the maximum packet length. (default). 1 Disable the zero length packet on transfers equal in length to a multiple maximum packet length.
28:27	Reserved. These bit reserved for future use and should be set to 0.
26:16	Maximum Packet Length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15	Interrupt On Setup (IOS). This bit is used on control type endpoints to indicate if USBINT is set in response to a setup being received.
14:0	Reserved. Bits reserved for future use and should be set to 0.

39.7.1.2 Transfer Overlay

The seven 32-bit words in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, it does not read the associated endpoint.

After an endpoint is readied, the dTD is copied into this queue head overlay area by the device controller. Until a transfer is expired, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller writes the results back to the original transfer descriptor and advances the queue.

See dTD for a description of the overlay fields.

39.7.1.3 Current dTD Pointer

The device controller uses the current dTD pointer to locate the transfer in progress. This word is for USB1 /USB0 (hardware) use only, and DCD software should not modified it.

Table 39-86. Current dTD Pointer

Bit	Description
31:5	Current dTD. This field is a pointer to the dTD that is represented in the transfer overlay area. This field is modified by the device controller to the next dTD pointer during endpoint priming or queue advance.
4:0	Reserved. Bit reserved for future use and should be set to zero.

39.7.1.4 Set-up Buffer

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID.

NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is for receiving setup data packets.

Table 39-87. Multiple Mode Control

32-bit word	Bits	Description
1	31:0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and the device controller writes it for software to read.
2	31:0	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and the device controller writes it for software to read.

39.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data sent/received for given transfer. The DCD should not attempt to modify any field in an active dTD except the next link pointer, which should only be modified as described in [Section 39.8.7, “Managing Transfers with Transfer Descriptors.”](#)

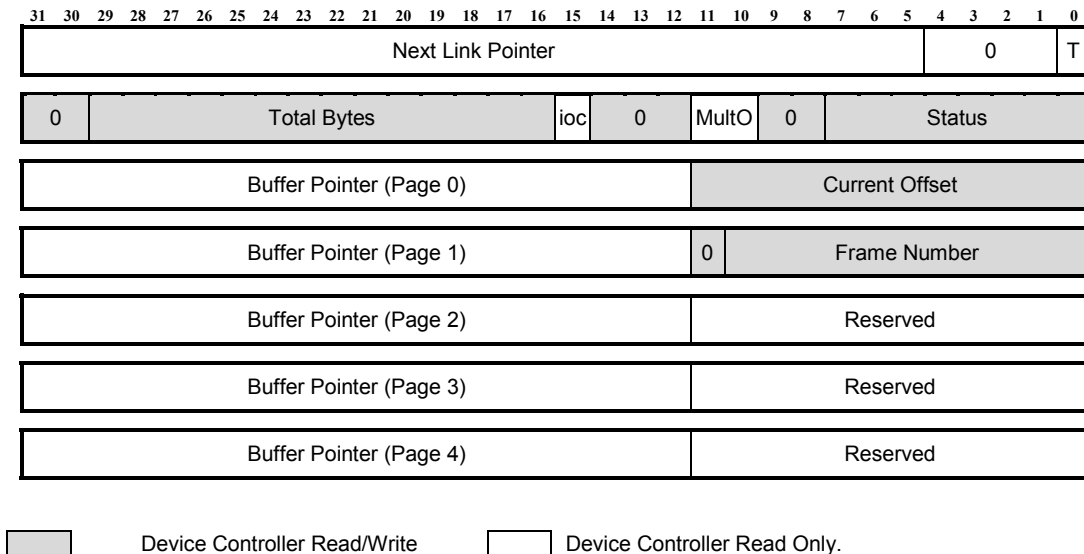


Figure 39-75. Endpoint Transfer Descriptor (dTD)

Table 39-88. Next dTD Pointer

Bit	Description
31:5	Next Transfer Element Pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively.

Table 39-88. Next dTD Pointer (continued)

Bit	Description
4:1	Reserved. Bits reserved for future use and should be set to 0.
0	Terminate (T). 1=pointer is invalid. 0=Pointer is valid (points to a valid Transfer Element Descriptor). This bit indicates to the device controller that no more valid entries exist in the queue.

Table 39-89. dTD Token

Bit	Description
31	Reserved. Bit reserved for future use and should be set to 0.
30:16	<p>Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction.</p> <p>The maximum value software may store in the field is 5*4K(5000H). This is the maximum number of bytes 5 page pointers can access. Although it is possible to create a transfer up to 20K this assumes the first offset into the first page is 0. When the offset cannot be predetermined, crossing passed the fifth page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K(4000H).</p> <p>If the value of the field is 0 when the host controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor.</p> <p>It is not a requirement for IN transfers that total bytes to transfer be an even multiple of maximum packet length. If software builds such a transfer descriptor for an IN transfer, the last transaction is always less than maximum packet length.</p>
15	Interrupt On Complete (IOC). This bit indicates if USBINT is to be set in response to device controller being finished with this dTD.
14:12	Reserved. Bits reserved for future use and should be set to 0.
11:10	<p>Multiplier Override (MultiO). This field can be used for transmit ISO's (i.e. ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO.</p> <p>Example:</p> <p>if QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 0 [default] Three packets are sent: {Data2(8); Data1(7); Data0(0)}</p> <p>if QH.multiplier = 3; Maximum packet size = 8; Total Bytes = 15; MultiO = 2 Two packets are sent: {Data1(8); Data0(7)}</p> <p>For maximal efficiency, software should compute MultiO = greatest integer of (Total Bytes/Max. Packet Size) except for the case when Total Bytes = 0; then MultiO should be 1.</p> <p>Note: Non-ISO and Non-TX endpoints must set MultiO=00.</p>
9:8	Reserved. Bits reserved for future use and should be set to zero.
7:0	<p>Status. This field is used by the Device Controller to communicate individual command execution states back to the Device Controller software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:</p> <p>Bit Status Field Description</p> <p>7 Active.</p> <p>6 Halted.</p> <p>5 Data Buffer Error.</p> <p>3 Transaction Error.</p> <p>4,2,0 Reserved.</p>

Table 39-90. Buffer Page Pointer List

Bit	Description
31:12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non-virtual memory systems typically set the buffer pointers to a series of incrementing integers.
0;11:0	Current Offset. Offset into the 4kb buffer where the packet is to begin.
1;10:0	Frame Number. Written by the device controller to indicate the frame number in which a packet finishes. This is typically used to correlate relative completion times of packets on an ISO endpoint.

39.8 Device Operational Model

The device operation transfers a request in the memory image to and from the Universal Serial Bus. Using a set of linked list transfer descriptors, pointed to by a queue head, the device controller performs the data transfers. The following sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

39.8.1 Device Controller Initialization

After hardware reset, the USB1 /USB0 is disabled until the run/stop bit is set to 1. In the disabled state, the pull-up on the USB D+ is not active, which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset occurs followed by setup packet arriving at endpoint 0. A queue head must be prepared so the device controller can store the incoming setup packet.

To initialize a device, software must perform these steps:

1. Set controller mode to device mode. Optionally set streaming disable in the USBMODE register.

NOTE

Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSC_n register if using a non-ULPI PHY.
4. Set iodic_b bit in the CONTROL register to enable PHY interface.
5. Allocate and Initialize device queue heads in system memory Minimum: Initialize device queue heads 0 Tx and 0 Rx.

NOTE

All device queue heads must be initialized for control endpoints before the endpoint is enabled. Device queue heads for non-control endpoints must be initialized before the endpoint can be used.

For information on device queue heads, refer to [Section 39.7, “Device Data Structures.”](#)

6. Configure ENDPOINTLISTADDR Pointer.

For additional information on ENDPOINTLISTADDR, refer to the register table.

7. Enable the microprocessor interrupt associated with the USB1 /USB0 and optionally change setting of ITC field in USBCMD register.

Recommended: Enable all device interrupts including: USBINT, USBERRINT, port change detect, USB reset received, and DCSuspend.

For a list of available interrupts, refer to the USBINTR and the USBSTS register tables.

8. Set run/stop bit to run mode.

After the run bit is set, a device reset occurs. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in the bus reset section of the following port state and control section below.

NOTE

Endpoint 0 is a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received is always a setup packet. The contents of the first setup packet requires a response in accordance with USB device framework command set.

39.8.2 Port State and Control

From a chip or system reset, the USB1 /USB0 enters the *powered* state. A transition from the powered state to the *attach* state occurs when the run/stop bit is set to 1. After receiving a reset on the bus, the port enters the *defaultFS or defaultHS* state in accordance with the protocol reset described in Appendix C.2 of the USB Specification Rev. 2.0. The following state diagram (Figure 39-76) depicts the state of a USB 2.0 device.

States powered, attach, defaultFS/HS, and suspendFS/HS are implemented in the USB1 /USB0 and are communicated to the DCD using these bits:

Table 39-91. Device Controller State Information Bits

Bit	Register
DCSuspend	USBSTS
USB Reset Received	USBSTS
Port Change Detect	USBSTS
High-Speed Port	PORTSC

It is the responsibility of the DCD to maintain a state variable to differentiate between the defaultfs/HS state and the address/configured states. Change of state from default to address and the configured states is part of the enumeration process described in the device framework section of the USB 2.0 Specification.

As a result of entering the address state, the DCD must program the device address register (DEVICEADDR).

Entry into the Configured indicates all endpoints are used in the operation of the device have been properly initialized by programming the ENDPTCTRL_n registers and initializing the associated queue heads.

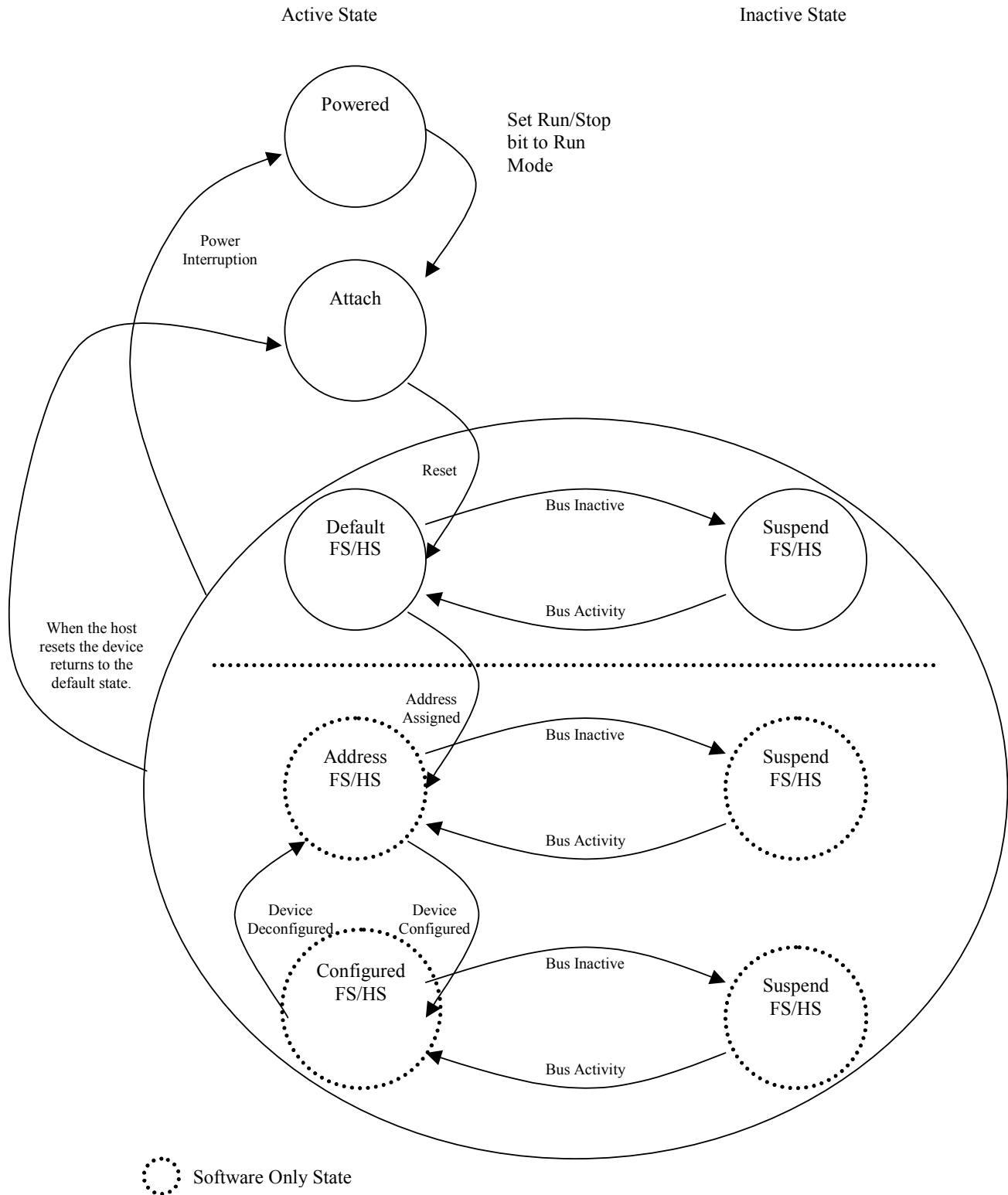


Figure 39-76. USB 2.0 Device States

39.8.3 Bus Reset

The host uses a bus reset to initialize downstream devices. When a bus reset is detected, the USB1 /USB0 controller renegotiates its attachment speed, resets the device address to 0, and notifies the DCD by interrupt (assuming the USB reset interrupt enable is set). After a reset is received, all endpoints (except endpoint 0) are disabled and the device controller cancels any primed transactions. The concept of priming is clarified below, but the DCD must perform the following tasks when a reset is received:

- Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.
- Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.
- Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF_FFFF to ENDPTFLUSH.

Read the reset bit in the PORTSC_n register and make sure that remains active. A USB reset occurs for a minimum of three ms and the DCD must reach this point in the reset cleanup before end of the reset occurs; otherwise, a hardware reset of the device controller is recommended (rare.)

Writing a one to the USB1 /USB0 reset bit in the USBCMD reset can perform a hardware reset.

NOTE

A hardware reset causes the device to detach from the bus by clearing the run/stop bit. Therefore, the DCD must completely re-initialize the USB1 /USB0 after a hardware reset.

Free all allocated dTDs because they are no longer executed by the device controller. If this is the first time the DCD is processing a USB reset event, it is likely that no dTDs have been allocated.

At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a port change detect is indicated.

After a port change detect, the device has reached the default state and the DCD can read the PORTSC_n to determine if the device is operating in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the USB Chapter 9 - Device Framework.

NOTE

The device DCD may use the FS/HS mode information to determine the bandwidth mode of the device.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

39.8.3.1 Suspend/Resume

39.8.3.1.1 Suspend Description

To conserve power, USB1 /USB0 automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB1 (/USB0 maintains any internal status, including its address and configuration. In device mode, the attached devices must be prepared to suspend at any time they are powered, regardless if they have been assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB1 /USB0 exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB1 /USB0 is capable of remote wake-up signaling. When the USB1 /USB0 is reset, remote wake-up signaling must be disabled.

39.8.3.1.2 Suspend Operational Model

The USB1 /USB0 moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, DCD is notified by an interrupt (assuming DC suspend interrupt is enabled). When the DCSuspend bit in the PORTSC n is set to a 1, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low-power operation.

Find information on the bus power limits in suspend state in USB 2.0 specification.

39.8.3.1.3 Resume

If the USB1 /USB0 is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB1 /USB0 can signal the system to resume operation by forcing resume signaling to the upstream port. Resume signaling is sent upstream by writing a '1' to the Resume bit in the in the PORTSC n while the device is in suspend state. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

NOTE

Before resume signaling can be used, the host must enable it by using the set feature command defined in device framework (chapter 9) of the USB 2.0 specification.

39.8.4 Managing Endpoints

The USB 2.0 specification defines an endpoint, also called a device endpoint or an address endpoint as a uniquely addressable portion of a USB device that can source or sink data in a communications channel between the host and the device. The endpoint address is specified by the combination of the endpoint number and the endpoint direction.

The channel between the host and an endpoint at a specific device represents a data pipe. Endpoint 0 for a device is always a control type data channel used for device discovery and enumeration. Other types of endpoints supported by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error handling. More detail on endpoint operation can be found in the USB 2.0 specification.

Both USB1 and USB0 support up to four (4) bidirectional endpoints, including the control endpoint. The DCD can enable, disable, and configure each endpoint.

Each endpoint direction is essentially independent and can be configured with differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0 is, for example, is always a control endpoint and uses the pair of directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum of four endpoint numbers, one for each endpoint direction the device controller uses, eight queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

39.8.4.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint zero are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to configuration bit in the `ENDPTCTRLn` register. Each 32-bit `ENDPTCTRLn` is split into an upper and lower half. The lower half of `ENDPTCTRLn` configures the receive or OUT endpoint and the upper half also configures the corresponding transmit or IN endpoint. Control endpoints must be configured the same in both the upper and lower half of the `ENDPTCTRLn` register otherwise the behavior is undefined. The following table shows how to construct a configuration word for endpoint initialization.

Table 39-92. Device Controller Endpoint Initialization

Field	Value
Data Toggle Reset	1
Data Toggle Inhibit	0
Endpoint Type	00 — Control 01 — Isochronous 10 — Bulk 11 — Interrupt
Endpoint Stall	0

39.8.4.2 Stalling

There are two occasions where the USB1 /USB0 may need to return to the host a STALL condition

The first occasion is the functional stall, which is a condition set by the DCD as described in the USB 2.0 device framework (chapter 9). A functional stall is used only on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the `ENDPTCTRLn` register associated with the given endpoint and the given direction. In a functional stall condition, the device controller continues to

return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.

A protocol stall, unlike a function stall, used on control endpoints is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the `ENDPTCTRLn` register can ensure both stall bits are set at the same instant.

NOTE

Any write to the `ENDPTCTRLn` register during operational mode must preserve the endpoint type field (i.e., perform a read-modify-write).

Table 39-93. Device Controller Stall Response Matrix

USB Packet	Endpoint Stall Bit.	Effect on STALL bit.	USB Response
SETUP packet received by a non-control endpoint.	N/A	None.	STALL
IN/OUT/PING packet received by a non-control endpoint.	1	None.	STALL
IN/OUT/PING packet received by a non-control endpoint.	0	None.	ACK/NAK/NYET
SETUP packet received by a control endpoint.	N/A	Cleared	ACK
IN/OUT/PING packet received by a control endpoint	1	None	STALL
IN/OUT/PING packet received by a control endpoint.	0	None.	ACK/NAK/NYET

39.8.4.3 Data Toggle

Data toggle is a mechanism to maintain data coherency between host and device for any given data pipe. For more information on data toggle, refer to the USB 2.0 specification.

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by writing a '1' to the data toggle reset bit in the `ENDPTCTRLn` register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

Data Toggle Inhibit is for test purposes only and should never be used during normal device controller operation.

Setting the data toggle inhibit bit active (1) causes the USB1 /USB0 to ignore the data toggle pattern normally sent and accept all incoming data packets regardless of the data toggle state.

In normal operation, the USB1 /USB0 checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If data PID does not match the data toggle state bit maintained by the device controller for that endpoint, the data toggle is considered not valid. If the data toggle is not valid, the device controller assumes the packet was already received and discards the packet (not reporting it to the DCD). To prevent the USB1 /USB0 from re-sending the same packet, the device controller responds to the error packet by acknowledging it with either an ACK or NYET response.

39.8.5 Device Operational Model For Packet Transfers

All transactions on the USB bus are initiated by the host and the device must respond to any request from the host within the turnaround time stated in the USB 2.0 Specification.

A USB host sends requests to the device controller (USB1 /USB0) in an order that cannot be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 3 (transmit direction) is configured as a bulk pipe, you can expect the host to send IN requests to that endpoint. The device controller prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as priming the endpoint. This term appears throughout the following documentation to describe the device controller (USB1 /USB0) operation so the DCD can be architected properly use priming. Further, the term flushing describes the action of clearing a packet queued for execution.

Priming a transmit endpoint causes the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it is stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to manage a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller.

After a priming request is complete, an endpoint state of primed is indicated in the ENDPTSTATUS register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of High Speed USB.

Because only the leading data is stored in the device controller FIFO, it is necessary for the device controller to begin filling in behind leading data after the transaction starts. The FIFO must be sized to account for the maximum latency that can be incurred by the system memory bus.

Priming receive endpoints are identical to priming of transmit endpoints from the point of view of the DCD. At the device controller, the major difference in the operational model is no data movement of the leading packet data simply because the data is to be received from the host.

As part of the architecture, the FIFO for the receive endpoints is not partitioned into multiple channels like the transmit FIFO. Therefore, the size of the RX FIFO does not scale with the number of endpoints.

39.8.5.1 Interrupt/Bulk Endpoint Operational Model

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes handshake with a NAK unless the endpoint had been primed. After the endpoint has been primed, data delivery commences.

A dTD is retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes *n* packets to be transferred according to the USB variable length transfer protocol. The formula and table on the following page describes how the device controller computes the

number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

$$N = \text{INT}(\text{Number Of Bytes}/\text{Max. Packet Length}) + 1$$

With Zero Length Termination (ZLT) = 1

$$N = \text{MAXINT}(\text{Number Of Bytes}/\text{Max. Packet Length})$$

Table 39-94. Variable Length Transfer Protocol Example (ZLT=0)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	
512	256	3	256	256	0
512	512	2	512	0	

Table 39-95. Variable Length Transfer Protocol Example (ZLT=1)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	
512	256	2	256	256	
512	512	1	512		

NOTE

The MULT field in the dQH must be set to 00 for bulk, interrupt, and control endpoints.

The ZLT bit in the dTD operates as following on BULK and control transfers:

ZLT = 0, the default value, means that the zero length termination is active. With the ZLT option enabled, when the device is transmitting, the hardware automatically appends a zero packet length when the following conditions are true:

- The packet transmitted equals maximum packet length.
- The dTD has exhausted the field Total Bytes

After this the dTD is retired. When the device is receiving, if the last packet length received equal maximum packet length and the total bytes is zero, it waits for a zero length packet from the host to retire the current dTD.

ZLT = 1, means the zero length termination is inactive. With the ZLT option disabled, when the device is transmitting, the hardware does not append any zero length packet. When receiving, it does not require a zero length packet to retire a dTD whose last packet was equal to the maximum packet length packet.

The dTD is retired as soon as Total Bytes field goes to zero, or a short packet is received.

Each transfer is defined by one dTD, so the zero length termination is for each dTD.

In some software application cases, the logic transfer does not fit into one dTD, so it does not make sense to add a Zero Length Termination packet each time a dTD is consumed. On those cases, it is recommended to turn off this ZLT feature and use software to generate the zero length termination.

TX-dTD is complete when:

- All packets described dTD were successfully transmitted. Total bytes in dTD equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. Total bytes in dTD equal zero when this occurs.
- A short packet (number of bytes < maximum packet length) was received. This is a successful transfer completion; DCD must check total bytes in dTD to determine the number of bytes that are remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) or (total bytes received > total bytes specified). This is an error condition. The device controller discards the remaining packet, and set the buffer error bit in the dTD. In addition, the endpoint is flushed and the USBERR interrupt becomes active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD is cleared and the next pointer is followed when the terminate bit is clear. When the terminate bit is set, the USB1 /USB0 flushes the endpoint/direction and ceases operations for that endpoint/direction.

On the unsuccessful completion of a packet (see long packet above), the dQH is left pointing to the dTD that was in error. To recover from this error condition, the DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

NOTE

All packet level errors, such as a missing handshake or CRC error, are retried automatically by the device controller.

There is no required interaction with the DCD for managing such errors.

Table 39-96. Interrupt/Bulk Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	Ignore	Ignore	Ignore	N/A	N/A
In	STALL	NAK	Transmit	BS Error ¹	N/A
Out	STALL	NAK	Receive + NYET/ACK ²	N/A	NAK
Ping	STALL	NAK	ACK	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

¹ Force Bit Stuff Error

- ² NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

NOTE

System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

39.8.5.2 Control Endpoint Operation Model

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase. The USB1 /USB0 always accepts the setup phase unless the setup lockout is engaged.

The setup lockout engages so future setup packets are ignored. Lockout of setup packets ensures that while software is reading the setup packet stored in the queue head, data is not written as it is being read potentially causing an invalid setup packet.

The setup lockout mechanism can be disabled and a tripwire type semaphore ensures the setup packet payload is extracted from the queue head without being corrupted by an incoming setup packet. This is preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling:

- Disable setup lockout by writing 1 to setup lockout mode (SLOM) in USBMODE. (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

NOTE

Leaving the setup lockout mode as 0 results in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
 - Write 1 to clear corresponding bit ENDPTSETUPSTAT.
 - Write 1 to setup TripWire (SUTW) in USBCMD register.
 - Duplicate contents of dQH.SetupBuffer into local software byte array.
 - Read setup TripWire (SUTW) in USBCMD register. (if set - continue; if cleared - goto 2)
 - Write 0 to clear setup Tripwire (SUTW) in USBCMD register.
 - Process setup packet using local software byte array copy and execute status/handshake phases.

NOTE

After receiving a new setup packet, the status and/or handshake phases may be pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

Following the setup phase, the DCD must create a device transfer descriptor for the data phase and prime the transfer.

After priming the packet, the DCD must verify a new setup packet has not been received by reading the ENDPTSETUPSTAT register immediately verifying that the prime had completed. A prime completes when the associated bit in the ENDPTPRIME register is 0 and the associated bit in the ENDPTSTATUS register is 1. If a prime fails, i.e. the ENDPTPRIME bit goes to zero and the ENDPTSTATUS bit is not set, then the prime has failed. This can only be due to improper setup of the dQH, dTD or a setup arriving during the prime operation. If a new setup packet is indicated after the ENDPTPRIME bit is cleared, the transfer descriptor can be freed and the DCD must reinterpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller automatically clears the prime status (ENDPTSTATUS) to enforce data coherency with the setup packet.

NOTE

The MULT field in the dQH must be set to 00 for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

Similar to the data phase, DCD must create a transfer descriptor (with byte length equal zero) and prime the endpoint for the status phase. The DCD must also perform the same checks of the ENDPTSETUPSTAT as described above in the data phase.

NOTE

The MULT field in the dQH must be set to 00 for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

Table 39-97 shows the device controller response to packets on a control endpoint according to the device controller state.

Table 39-97. Control Endpoint Bus Response Matrix

Token Type	Endpoint State						Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	Not Enabled	
Setup	ACK	ACK	ACK	N/A	SYSERR ¹	BTO	—
In	STALL	NAK	Transmit	BS Error ²	N/A	BTO	N/A
Out	STALL	NAK	Receive + NYET/ACK ³	N/A	NAK	BTO	N/A
Ping	STALL	NAK	ACK	N/A	N/A	BTO	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	BTO	Ignore

¹ SYSERR — System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

² Force Bit Stuff Error

- ³ NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

39.8.5.3 Isochronous Endpoint Operational Model

Isochronous endpoints are used for real-time scheduled delivery of data and their operational model is significantly different than the host throttled bulk, interrupt, and control data pipes. Real time delivery by the USB1 /USB0 is accomplished by:

- Exactly MULT Packets per (micro) Frame are transmitted/received.

NOTE

MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.

- NAK responses are not used. Instead, zero length packets are sent in response to an IN request to an unprimed endpoint. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If the ISO-dTD remains active after that frame, the ISO-dTD is held ready until executed or canceled by the DCD.

The USB1 /USB0 in host mode uses the periodic frame list to schedule data exchanges to isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for control/bulk/interrupt endpoints is also for isochronous endpoints. The difference is in the handling of the dTD.

The first difference between bulk and ISO-endpoints is priming an ISO-endpoint is a delayed operation such that an endpoint becomes primed only after a SOF is received. After the DCD writes the prime bit, the prime bit is cleared as usual to indicate to software the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that prime start until the next frame boundary. This behavior is hidden from the DCD, but occurs so the device controller can match the dTD to a specific (micro)frame.

Another difference with isochronous endpoints is the transaction must wholly complete in a (micro)frame. After an ISO transaction is started in a (micro)frame, it retires the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, the device controller forces retirement of the ISO-dTD and move to the next ISO-dTD.

Fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction stays primed indefinitely. This means it is up to software to discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

The last difference with ISO packets is in the data level error handling. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the transaction error bit and the data is stored as usual for the application software to sort out.

- TX Packet Retired

- MULT counter reaches zero.
- Fulfillment Error [Transaction Error bit is set]
- #Packets Occurred > 0 AND # Packets Occurred < MULT

NOTE

For TX-ISO, MULT counter can be loaded with a lesser value in the dTD multiplier override field. If the multiplier override is zero, the MULT counter is initialized to the Multiplier in the QH.

- RX Packet Retired:
 - MULT counter reaches zero.
 - Non-MDATA Data PID is received
 - Overflow Error:
 - Packet received is > maximum packet length. [Buffer Error bit is set]
 - Packet received exceeds total bytes allocated in dTD. [Buffer Error bit is set]
 - Fulfillment Error [Transaction Error bit is set]
 - # Packets Occurred > 0 AND # Packets Occurred < MULT
 - CRC Error [Transaction Error bit is set]

NOTE

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame-to-(micro)frame operation, DCD should ensure the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

39.8.5.3.1 Isochronous Pipe Synchronization

When necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can act as a marker. To cause a packet transfer to occur at a specific (micro)frame number [N], DCD should interrupt on SOF during frame N-1. When the FRINDEX equals N-1, the DCD must write the prime bit. The USB1/USB0 primes the isochronous endpoint in (micro)frame N-1 so that the device controller executes delivery during (micro)frame N.

CAUTION

Priming an endpoint towards the end of (micro)frame N-1 does not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if device controller does not have enough time to complete the prime before the SOF for packet N is received.

Table 39-98. Isochronous Endpoint Bus Response Matrix

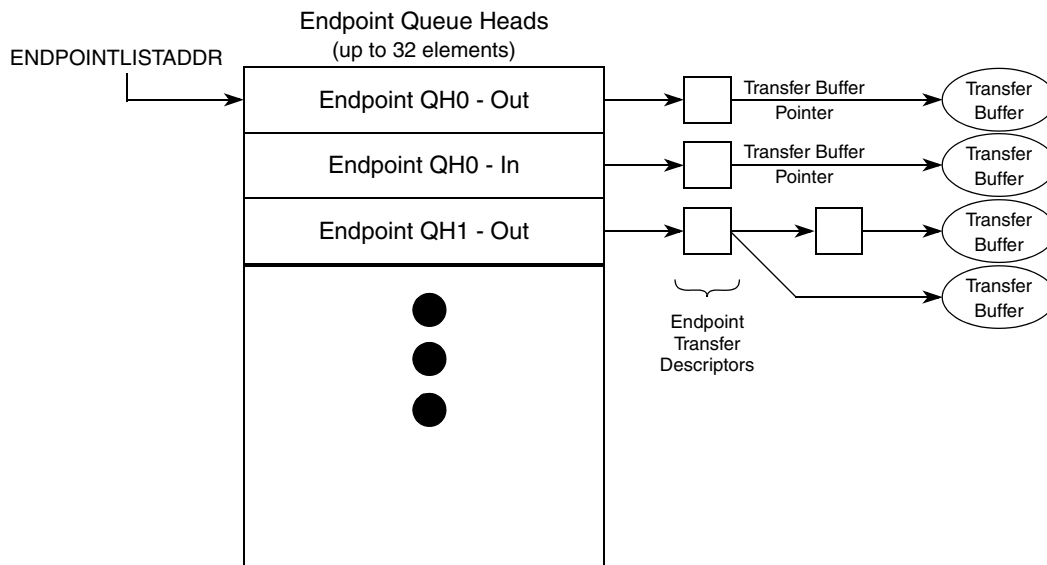
	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL ¹ Packet	NULL Packet	Transmit	BS Error ²	N/A

Table 39-98. Isochronous Endpoint Bus Response Matrix (continued)

	Stall	Not Primed	Primed	Underflow	Overflow
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

¹ Zero Length Packet² Force Bit Stuff Error

39.8.6 Managing Queue Heads

**Figure 39-77. Endpoint Queue Head Diagram**

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device transfer descriptor (dTD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQH's in a sequential list as shown in Figure 39-77. The even elements in the list of dQH's are receive endpoints (OUT/SETUP) and the odd elements are for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. After the dTD has been retired, it is no longer part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors because pointers no longer exist within the queue head once the dTD is retired (see section Software Link Pointers).

In addition to the current and next pointers and the dTD overlay examined in the section on operational model for packet transfers, dQH also contains the following parameters for the associated endpoint: multiplier, maximum packet length, interrupt on setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

39.8.6.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB Chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1, 2, or 3 as required bandwidth with the USB Chapter 9 protocol.

NOTE

In FS mode, the multiplier field can only be 1 for ISO endpoints.

- Write the next dTD Terminate bit field to 1.
- Write the active bit in the status field to 0.
- Write the halt bit in the status field to 0.

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTD's.

39.8.6.2 Operational Model For Setup Transfers

As discussed in [Section 39.8.5.2, “Control Endpoint Operation Model,”](#) setup transfer requires special treatment by the DCD. A setup transfer does not use a dTD, but stores the incoming data from a setup packet in an 8-byte buffer within the dQH instead.

Upon receiving notification of the setup packet, DCD should manage setup transfer as demonstrated here:

1. Copy setup buffer contents from dQH - RX to software buffer.
2. Acknowledge setup backup by writing a 1 to the corresponding bit in ENDPTSETUPSTAT.

NOTE

The acknowledge must occur before continuing to process the setup packet.

NOTE

After the acknowledge has occurred, the DCD must not attempt to access the setup buffer in the dQH - RX. Only the local software copy should be examined.

3. Check for pending data or status dTDs from previous control transfers and flush if any exist as discussed in section flushing/de-priming an endpoint.

NOTE

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

39.8.7 Managing Transfers with Transfer Descriptors

39.8.7.0.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers for the linked list of dTDs for each respective queue head. This is necessary because the dQH only maintains pointers to the current working dTD and the next dTD to be executed. The operations described in the next section for managing dTD assumes the DCD can use reference for the head and tail of the dTD linked list.

NOTE

To conserve memory, reserved fields at the end of the dQH can store the head and tail pointers, but it remains the responsibility of the DCD to maintain the pointers.

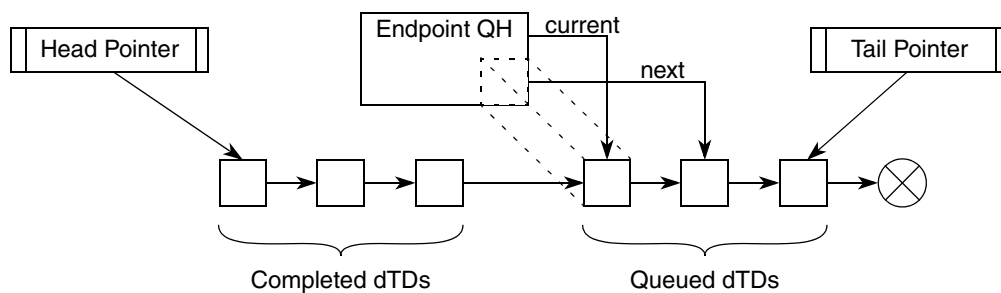


Figure 39-78. Software Link Pointers

39.8.7.1 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate 8 32-bit word dTD block of memory aligned to 8 32-bit word boundaries. Example: bit address 4:0 would be equal to 00000.

Write the following fields:

1. Initialize first 7 32-bit words to 0.
2. Set the terminate bit to 1.
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to 1 and all remaining status bits set to 0.
6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointer.

39.8.7.2 Executing A Transfer Descriptor

To safely add a dTD, DCD must follow this procedure which manages the event where the device controller reaches the end of the dTD list at the same time a new dTD is added to the end of the list.

Determine whether the link list is empty:

Check DCD driver to see if pipe is empty (internal representation of linked-list should indicate if any packets are outstanding)

Case 1: Link list is empty

8. Write dQH next pointer AND dQH terminate bit to 0 as a single 32-bit word operation.
9. Clear active and halt bit in dQH (in case set from a previous error).
10. Prime endpoint by writing 1 to correct bit position in ENDPTPRIME.

Case 2: Link list is not empty

1. Add dTD to end of linked list.
2. Read correct prime bit in ENDPTPRIME - if 1 DONE.
3. Set ATDTW bit in USBCMD register to 1.
4. Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later)
5. Read ATDTW bit in USBCMD register.
 - If 0, goto 3.
 - If 1, continue to 6.
6. Write ATDTW bit in USBCMD register to '0'.
7. If status bit read in (3) is '1' DONE.
8. If status bit read in (3) is '0' then Goto Case 1: Step 1.

39.8.7.3 Transfer Completion

After a dTD has been initialized and the associated endpoint primed, the device controller executes the transfer upon the host-initiated request. The DCD is notified with a USB interrupt if the interrupt on complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

CAUTION

Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs have finished (active bit cleared).

By reading the status fields of the completed dTDs, DCD can determine if the transfers completed successfully. Success is determined with this combination of status bits:

- Active = 0
- Halted = 0
- Transaction Error = 0

- Data Buffer Error = 0

Should any combination other than the one shown above exist, DCD must take proper action. Transfer failure mechanisms are indicated in the device error matrix.

In addition to checking the status bit, DCD must read the transfer bytes field to determine the actual bytes transferred. When a transfer is complete, the total bytes transferred is decremented by the actual bytes transferred. For transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according to the USB variable length packet protocol.

39.8.7.4 Flushing/De-priming an Endpoint

It is necessary for the DCD to flush to de-prime one more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The DCD can use the following procedure to stop a transfer in progress:

1. Write a 1 to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are 0.

NOTE

This operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.

3. Read ENDPTSTATUS to ensure all endpoints commanded to be flushed are now 0. If the corresponding bits are 1 after the second step has finished, flush has failed as described here:

In rare cases, a packet is in progress to the particular endpoint is commanded to flush using ENDPTFLUSH. A safeguard is in place to refuse the flush to ensure the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush by repeating steps one through three until each endpoint is successfully flushed.

39.8.8 Device Error Matrix

The following table summarizes packet errors that are not automatically managed by the USB1 /USB0.

Table 39-99. Device Error Matrix

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Overflow ¹	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

¹ This error also set the halt bit in the dQH and if there are dTDs remaining in the linked list for the endpoint, those are not executed.

The device controller manages all errors on bulk/control/interrupt endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated.

Table 39-100. Error Descriptions

Overflow	Number of bytes received exceeded max. packet size or total buffer length.
ISO Packet Error	CRC Error on received ISO packet. Contents not guaranteed to be correct.
ISO Fulfillment Error	Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery, DCD may need to readjust the data queue because a fulfillment error causes device controller to cease data transfers on the pipe for one (micro)frame. During dead (micro)frame, device controller reports error on the pipe and primes for the next frame.

39.8.9 Servicing Interrupts

The interrupt service routine must consider there are high-frequency, low-frequency, and error operations to order accordingly.

39.8.9.1 High-Frequency Interrupts

High frequency interrupts in particular should be managed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

Table 39-101. Interrupt Handling Order

Execution Order	Interrupt	Action
1a	USB Interrupt ¹ ENDPTSETUPSTATUS	Copy contents of setup buffer and acknowledge setup packet (as indicated in section Managing Queue Heads). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.
1b	USB Interrupt ENDPTCOMPLETE	Manages completion of dTD as indicated in section Managing Queue Heads.
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

¹ It is likely that multiple interrupts to stack up on any call to the interrupt service routine and during the interrupt service routine.

39.8.9.2 Low-Frequency Interrupts

The low-frequency events include the following interrupts. These interrupts can be managed in any order because they don't occur often in comparison to the high-frequency interrupts.

Table 39-102. Low Frequency Interrupt Events

Interrupt	Action
Port Change	Change software state information.
Sleep Enable (Suspend)	Change software state information. Low power handling as necessary.
Reset Received	Change software state information. Abort pending transfers.

39.8.9.3 Error Interrupts

Error interrupts are least frequent and should be placed last in the interrupt service routine.

Table 39-103. Error Interrupt Events

Interrupt	Action
USB Error Interrupt.	This error is redundant because it combines USB Interrupt and an error status in the dTD. The DCD more aptly manages packet-level errors by checking dTD status field upon receipt of USB Interrupt (w/ ENDPTCOMPLETE).
System Error	Unrecoverable error. Immediate Reset of core; free transfers buffers in progress and restart the DCD.

39.8.10 Deviations from the EHCI Specifications

The host mode operation of the modules is nearly EHCI-compatible with few minor differences. For the most part, the modules conform to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in these areas:

- Device operation (OTG module only)—In host mode, the device operational registers are generally disabled and device mode is mostly transparent when in host mode. However, there are exceptions documented in the following sections.
- Embedded design interface—The modules do not have a PCI interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

For the purposes of the USB1 /USB0 implementing dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device operation and OTG operation are not specified in the EHCI, and implementation supported in the OTG module is proprietary.

39.8.10.1 Device Operation

The co-existence of a device operational controller within the OTG module has little effect on EHCI compatibility for host operation. However, because the OTG controller is initialized in neither host nor device mode, the USBMODE register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

39.8.10.2 Non-Zero Fields in the Register File

Some of the reserved fields and reserved addresses in the capability registers and operational registers have meaning in device mode; therefore, the following must be adhered to:

- Write operations to all EHCI reserved fields (some of which are device fields in the OTG module) in the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the module must properly mask EHCI reserved fields (some of which are device fields in the OTG module registers).

39.8.10.3 SOF Interrupt

The SOF interrupt is a free running 125 μ sec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. The free running interrupt is shared with the OTG-only device-mode start-of-frame interrupt. See [Section 39.2.1.4.2, “USB Status](#)

Register (USBSTS),” and Section 39.2.1.4.3, “USB Interrupt Enable Register (USBINTR),” for more information.

39.8.10.4 Embedded Design

This is an embedded USB host controller as defined by the EHCI specification and does not implement the PCI configuration registers.

39.8.10.4.1 Frame Adjust Register

Because the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the frame adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125 μ sec using the transceiver clock as a reference clock. In other words, 60 MHz transceiver clock is used for 8-bit physical interfaces and full-speed serial interfaces or 30 a MHz transceiver clock is used for 16-bit physical interfaces.

39.8.10.5 Miscellaneous Variations from EHCI

39.8.10.5.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces which can operate in different modes when the module is configured with the software programmable physical interface modes. The control bits for selecting the PHY operating mode have been added to the PORTSC_n register providing a capability defined by EHCI specification.

39.8.10.5.2 Discovery

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed, counters are already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall write a 1 to the reset bit of the device.
- Software shall write a 0 to the reset bit of the device after 10 msec.
 - This step (necessary in a standard EHCI design) may be omitted with this implementation. Should the EHCI host controller driver attempt to write a 0 to the reset bit while a reset is in progress, the write is ignored and the reset continues until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller the device is now operational, and at this point, the port speed has been determined.

After the port change interrupt indicates a port is enabled, EHCI stack should determine the port speed. Unlike EHCI implementation, which re-assigns the port owner for any device that does not connect at high-speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner is read-only and always reads 0.
- A 2-bit port speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit high speed indicator has been added to PORTSC to signify that the port is in HS vs. FS/LS
 - This information is redundant with the 2-bit port speed indicator above.

39.9 USB 2.0 PHY with On-The-Go

39.9.1 Introduction

The USB2 PHY is a fully integrated PHY core with High Speed(HS), Full Speed(FS) and Low Speed(LS) transceivers compliant with the USB 2.0 and USB 1.1 specifications. The front end of the USB2 PHY is the cable. The back end of the USB2 PHY is the serial/parallel interface engine that manages packet recognition, transaction sequencing, serialization/deserialization, bit stuffing/unstuffing and other relevant functionality needed by USB 2.0 specification. The USB2 transceiver's main responsibility is to transmit data onto the line, receive data, and to recover clock correctly from the received data.

39.9.1.1 Features

- USB2 PHY
 - HS Driver: Generates 400mV(approx.) logic levels on DP/DM at 480 MHz.
 - LS/FS Driver: It is used for low/full-speed data transmission at 12 MHz.
 - HS Receiver: This HS receiver receives the high-speed (480 Mb/s) NRZI differential data signal from the USB port and converts it to single ended signal.
 - LS/FS Receiver: The low/full-speed receiver receives the full speed (12 Mb/s) or low-speed(1.5 Mb/s) NRZI differential data signal from the USB port and convert it to single ended signal.
 - Transmission envelope detector: Monitors the differential amplitude of the data signal and alerts the controller if the differential signal goes below the specifications.
 - Disconnection Envelope Detector: Monitors the amplitude of the high-speed differential data signal and alerts the controller if the amplitude exceeds the specifications.
 - PLL: PLL requires a 24 MHz crystal clock as a reference and produces 480 MHz clock for the HS DLL and HS TX.
 - Single Ended Receiver: Monitors the amplitude of the FS/LS data signal at each DP and DM and alerts the controller if the single-ended amplitude is less than specifications.
 - Termination Resistor Calibration: During high-speed transfers, data line resistor terminations are required to be within 45 ohms +/- 10%. To achieve this accuracy, these termination resistors are calibrated to the precise off chip resistance.
 - Xtal Osc: Provides the 24 MHz low jitter clock to the USB 480 MHz PLL.
- On-The-Go (in addition to the above blocks)
 - VBUS Comparator: Monitors the VBUS and generates output corresponding to VBUS valid and session valid during SRP and HNP.

- Switches and Resistors: To act as a dual role device, RPU and RPD need to be switched depending on whether a host or device. In OTG mode, certain bus resistances need to be switched to meet the requirements for SRP and HNP.
- ID Detector: Depending upon the status of the ID pin, this block determines the allocated default status whether OTG host/device.

For more description on the above blocks refer to *USB Rev2.0 Specification* and *On-The-Go Supplement to the USB 2.0 Specification rev1.0a*.

39.9.1.2 Modes of Operation

- Low Speed/Full speed TX/RX: In this mode, data is transmitted/received serially at 1.5 Mbps/12 Mbps. For more details, refer to USB rev2.0 Specification document.
- High Speed TX/RX: In this mode, data is transmitted/received serially at 480 Mbps. For more details, refer to USB Rev2.0 Specification document.
- USB On-The-Go: This mode adds the capability of communication between one battery powered device to another via USB interface. For more details, refer to *On-The-Go Supplement to the USB 2.0 Specification rev1.0a*.

39.9.1.3 System Requirements

- External VBUS Generator: The USB PHY with OTG does not have an on-chip charge pump; therefore, it requires the use of an off-chip VBUS generator to source up to 500mA at 5V. The MAX1838 or a comparable part should be sufficient.
- External Resistor Divider: The VBUS pin for monitoring VBUS requires the voltage be stepped down from 5V to TBDV.

Chapter 40

Video-In (VIU)

40.1 Introduction

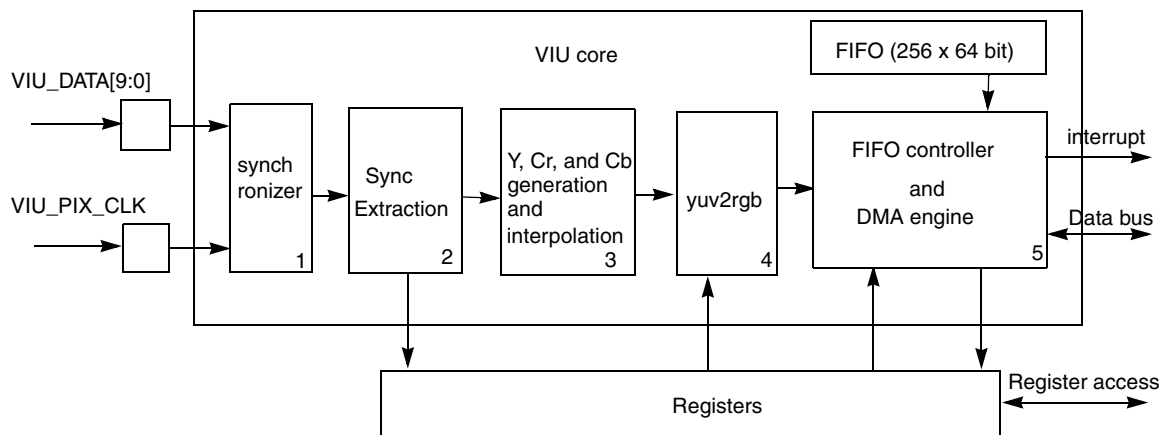


Figure 40-1. Video In Block Diagram

A block diagram of the VIDEO_IN is given in [Figure 40-1](#). There are two main components in it: VIU core and register.

VIU core accepts an ITU656 compatible video stream on its parallel interface. Internal synchronizer block latches data from pixel clock domain to internal working clock domain, then the synchronization extraction block extracts video timing signals such as hsync and vsync. Depending on timing signals, VIU can implement the following processing functions. The first processing function is converting the 422 stream to a 444 stream by providing interpolation on the chroma components, but it keeps YCbCr format. The second processing function is converting YCbCr stream to RGB888/RGB565 format.

Block 5 functions as FIFO controller and DMA engine. It puts the RGB pixel data into a FIFO and writes them to memory finally by the embedded DMA.

Register component acts as VIU core controller. CPU can totally control VIU core via configuring registers in register block.

40.1.1 Features

- Support from QVGA to XVGA 8-bit/10-bit ITU656 video input
- YUV to RGB888/565 conversion
- De-interlace for interlacing video image by setting DMA_INCREMENT register
- Internal DMA engine for data transferring from FIFO to memory

40.2 External Signal Description

Table 40-1. External Signal Properties

Name	Port	Function	I/O
VIU_PIX_CLK		ITU656 video stream clock	I
VIU_DATA[9:0]		ITU656 video stream data	I

40.3 Memory Map and Register Definition

40.3.1 Memory Map

Table 40-2. VIDEO_IN Memory Map

Offset or Address	Register	Access	Section/Page
0x00	Status and Configuration Register	RW	40.3.3.1/40-5
0x04	Luminance Coefficients For Red, Green and Blue Matrix	RW	40.3.3.2/40-7
0x08	Chroma Coefficients For Red Matrix	RW	40.3.3.3/40-8
0x0C	Chroma Coefficients For Green Matrix	RW	40.3.3.4/40-8
0x10	Chroma Coefficients For Blue Matrix	RW	40.3.3.5/40-9
0x14	Base Address Of Every Field Of Picture In Memory	RW	40.3.3.6/40-10
0x18	Horizontal Dma Increment	RW	40.3.3.7/40-10
0x1C	Max Pixel and Line Count	RW	40.3.3.8/40-11
0x20	High Priority Transfer Request Alarm	RW	40.3.3.9/40-12
0x24	Programable Alpha Value	RW	40.3.3.10/40-12

40.3.2 Register Summary

[Table 17-5](#) shows the VIDEO_IN register summary table.

Table 40-3. VIDEO_IN Register Summary

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X00 STATUS_CONFIG	R	MOD E_32 BIT	ROU ND_ ON	DITH ER_ ON	FIEL D_N O	DMA _AC T						ERR OR_I RQ	DMA _EN D_IR Q	VST ART _IRQ	HSY NC_I RQ	VSY NC_I RQ	FIEL D_IR Q
	W																
	R		ECC _EN	ERR OR_ EN	DMA _EN D_E N	VST ART _EN	HSY NC_ EN	VSY NC_ EN	FIEL D_E N	ERROR_CODE							SOF T_R ESE T
	W																
0X04 LUMINANCE_COE FFICIENTS	R	Y_RED[9:0]											Y_GREEN[9:5]				
	W																
	R	Y_GREEN[4:0]						Y_BLUE[9:0]									
	W																
0X08 RED_CHROMA_C OEFFICIENTS	R						CR_RED[10:0]										
	W																
	R						CB_RED[11:0]										
	W																
0X0C GREEN_CHROMA _COEFFICIENTS	R						CR_GREEN[10:0]										
	W																
	R						CB_GREEN[11:0]										
	W																
0X10 BLUE_CHROMA_ COEFFICIENTS	R						CR_BLUE[10:0]										
	W																
	R						CB_BLUE[11:0]										
	W																
0X14 DMA_ADDRESS	R	DMA_ADDRESS[31:16]															
	W																
	R	DMA_ADDRESS[15:3]															
	W																
0X18 DMA_INCREMEN T	R																
	W																
	R	DMA_INCREMENT[15:3]															
	W																

Table 40-3. VIDEO_IN Register Summary (continued)

Name		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0X1C PICTURE_COUNT	R	LINE_COUNT[15:0]															
	W																
	R	PIXEL_COUNT[15:0]															
	W																
0X20 HI_PRIO_ALARM	R																
	W																
	R	HI_PRIO_ALARM[15:0]															
	W																
0X24 ALPHA	R																
	W																
	R									ALPHA[7:0]							
	W																

40.3.3 Register Descriptions

40.3.3.1 STATUS_CONFIG

Offset 0x00				Access: User read/write				
Power	0	1	2	3	4	5	6	7
Architecture								
conventional	31	30	29	28	27	26	25	24
R	MODE_32BIT	ROUND_ON	DITHER_ON	FIELD_NO	DMA_ACT			
W								
Reset	0	0	0	0	0	0	0	0
Power	8	9	10	11	12	13	14	15
Architecture								
conventional	23	22	21	20	19	18	17	16
R			ERROR_IRQ	DMA_END_IRQ	VSTART_IRQ	HSYNC_IRQ	VSYNC_IRQ	FIELD_IRQ
W								
Reset	0	0	0	0	0	0	0	0
Power	16	17	18	19	20	21	22	23
Architecture								
conventional	15	14	13	12	11	10	9	8
R		ECC_EN	ERROR_EN	DMA_END_EN	VSTART_EN	HSYNC_EN	VSYNC_EN	FIELD_EN
W								
Reset	0	0	0	0	0	0	0	0
Power	24	25	26	27	28	29	30	31
Architecture								
conventional	7	6	5	4	3	2	1	0
R	ERROR_CODE							SOFT_RESET
W								
Reset	0	0	0	0	0	0	0	0
	= Unimplemented or Reserved							

Table 40-4. STATUS_CONFIG Register

Table 40-5. STATUS_CONFIG Fields

Field	Description
SOFT_RESET	Writing 1 to this bit generates an internal reset to all components except registers in VIDEO_IN block. This bit should be set by software when core receives an error interrupt.
ERROR_CODE	Error code. Signals error that triggered error IRQ. 0000 : No error 0001 : DMA arm command given during vertical active, DMA_ACT does not accept the value on IPS bus. 0010 : DMA arm command given during vertical blanking when DMA_ACT is set. 0100 : Line too long 0101 : Too many lines in field 0110: Line too short 0111: Not enough lines in field 1000: FIFO overflow 1001 : FIFO underflow 1010: One bit ECC error 1011: Two or more bits ECC error
FIELD_EN	Interrupt enable bit for FIELD_IRQ.
VSYNC_EN	Interrupt enable bit for VSYNC_IRQ.
HSYNC_EN	Interrupt enable bit for HSYNC_IRQ.
VSTART_EN	Interrupt enable bit for VSTART_IRQ.
DMA_END_EN	Interrupt enable bit for DMA_END_IRQ.
ERROR_EN	Interrupt enable bit for ERROR_IRQ.
ECC_EN	Enable bit for ECC.
FIELD_IRQ	Interrupt status bit. Set when field number is changed. Write 1 to clear FIELD_IRQ.
VSYNC_IRQ	Interrupt status bit. Set when one active field is over on ITU interface. Write 1 to clear VSYNC_IRQ.
HSYNC_IRQ	Interrupt status bit. Set when one active line is over on ITU interface. Write 1 to clear HSYNC_IRQ.
VSTART_IRQ	Interrupt status bit. Set when one active field starts on ITU interface. Write 1 to clear VSTART_IRQ.
DMA_END_IRQ	Interrupt status bit. Set when one active field data transfer is done. Write 1 to clear DMA_END_IRQ.
ERROR_IRQ	Interrupt status bit. Set when any error event occurs. Write 1 to clear ERROR_IRQ.
DMA_ACT	Enable DMA transfer or DMA transfer of current frame is busy when it is set. Write by software, cleared at end of transfer. When DMA_ACT is cleared, input video data is ignored and not put into FIFO.
FIELD_NO	Field number, extracted from ITU-656 stream.
DITHER_ON	Dithering is on. Used when video data is stored in buffer as RGB565 format and ROUND_ON is not set.
ROUND_ON	Round is on. Used when video data is stored in buffer as RGB565 format.
MODE_32BIT	If this bit is set, data is written to memory in RGBa888 format. DITHER_ON and ROUND_ON is ignored. If this bit is clear, data is written to memory in RGB565 format.

40.3.3.2 LUMINANCE_COEFFICIENTS

offset 0x04																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	Y_RED[9:0]											Y_GREEN[9:5]					
W																	
Reset	1	0	0	1	0	1	0	1	0	0	0	1	0	0	1	0	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	Y_GREEN[4:0]						Y_BLUE[9:0]										
W																	
Reset	1	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	


 = Unimplemented or Reserved

Figure 40-2. LUMINANCE_COEFFICIENTS Register

Table 40-6. LUMINANCE_COEFFICIENTS Fields

Field	Description
Y_RED[9:0]	Luminance coefficient for red matrix.
Y_GREEN[9:0]	Luminance coefficient for green matrix.
Y_BLUE[9:0]	Luminance coefficient for blue matrix.

The RGB pixel value is computed using following formulae:

$$\text{Red} = \frac{(Y - 16) \cdot y_{\text{red}}}{512} + \frac{(Cr - 128)Cr_{\text{red}}}{512} + \frac{(Cb - 128)Cb_{\text{red}}}{512}$$

$$\text{Green} = \frac{(Y - 16) \cdot Y_{\text{green}}}{512} + \frac{(Cr - 128)Cr_{\text{green}}}{512} + \frac{(Cb - 128)Cb_{\text{green}}}{512}$$

$$\text{Blue} = \frac{(Y - 16) \cdot y_{\text{blue}}}{512} + \frac{(Cr - 128)Cr_{\text{blue}}}{512} + \frac{(Cb - 128)Cb_{\text{blue}}}{512}$$

The multiplications with Y_red, Y_green, and Y_blue are unsigned multiplications. The multiplications with Cr_red, Cb_red, Cr_green, Cb_green, Cr_blue, and Cb_blue are signed multiplications. The addition is saturated to prevent overflow.

40.3.3.3 RED_CHROMA_COEFFICIENTS

offset 0x08																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R						CR_RED[10:0]											
W																	
Reset	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R					CB_RED[11:0]												
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 40-3. RED_CHROMA_COEFFICIENTS Register

Table 40-7. RED_CHROMA_COEFFICIENTS Fields

Field	Description
CR_RED[10:0]	Cr coefficient for red matrix.
CB_RED[11:0]	Cb coefficient for red matrix.

40.3.3.4 GREEN_CHROMA_COEFFICIENTS

offset 0x0C																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R						CR_GREEN[10:0]											
W																	
Reset	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R					CB_GREEN[11:0]												
W																	
Reset	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	

= Unimplemented or Reserved

Figure 40-4. GREEN_CHROMA_COEFFICIENTS Register

Table 40-8. GREEN_CHROMA_COEFFICIENTS Fields

Field	Description
CR_GREEN[10:0]	Cr coefficient for green matrix.
CB_GREEN[11:0]	Cb coefficient for green matrix.

40.3.3.5 BLUE_CHROMA_COEFFICIENTS

offset 0x10													read/write			
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R						CR_BLUE[10:0]										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R					CB_BLUE[11:0]											
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1
		= Unimplemented or Reserved														

Figure 40-5. BLUE_CHROMA_COEFFICIENTS Register

Table 40-9. BLUE_CHROMA_COEFFICIENTS Fields

Field	Description
CR_BLUE[10:0]	Cr coefficient for blue matrix.
CB_BLUE[11:0]	Cb coefficient for blue matrix.

40.3.3.6 DMA_ADDRESS

offset 0x14																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	DMA_ADDRESS[31:16]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	DMA_ADDRESS[15:3]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 40-6. DMA_ADDRESS Register

Table 40-10. DMA_ADDRESS Fields

Field	Description
DMA_ADDRESS[31:3]	Base address of every field of picture in memory used by DMA. Rewrite only after receiving DMA end interrupt and before arming DMA. The lowest 3 bits of DMA_ADDRESS cannot be set. It is always 3'b0. See section 40.5.2/40-17 for more details.

40.3.3.7 DMA_INCREMENT

offset 0x18																read/write	
Power Architecture	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Conventional	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R																	
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Power Architecture	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Conventional	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	DMA_INCREMENT[15:3]																
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

= Unimplemented or Reserved

Figure 40-7. DMA_INCREMENT Register

Table 40-11. DMA_INCREMENT Fields

Field	Description
DMA_INCREMENT [15:3]	Value of this field should be zero or memory size that one active line occupies in memory. It is added to the memory mapped rounded address at the end of every line. See section 40.4.3/40-15. Memory size of one active line depends on line pixel number(See section 40.3.3.8/40-11). It is <ul style="list-style-type: none"> • PIXEL_COUNT[15:2] + IPIXEL_COUNT[1:0] when MODE_32BIT=0; • PIXEL_COUNT[15:1] + PIXEL_COUNT[0] when MODE_32BIT=1; It is only configured when DMA is non-active, during vertical blanking. See section 40.5.2/40-17 for more details.

40.3.3.8 PICTURE_COUNT

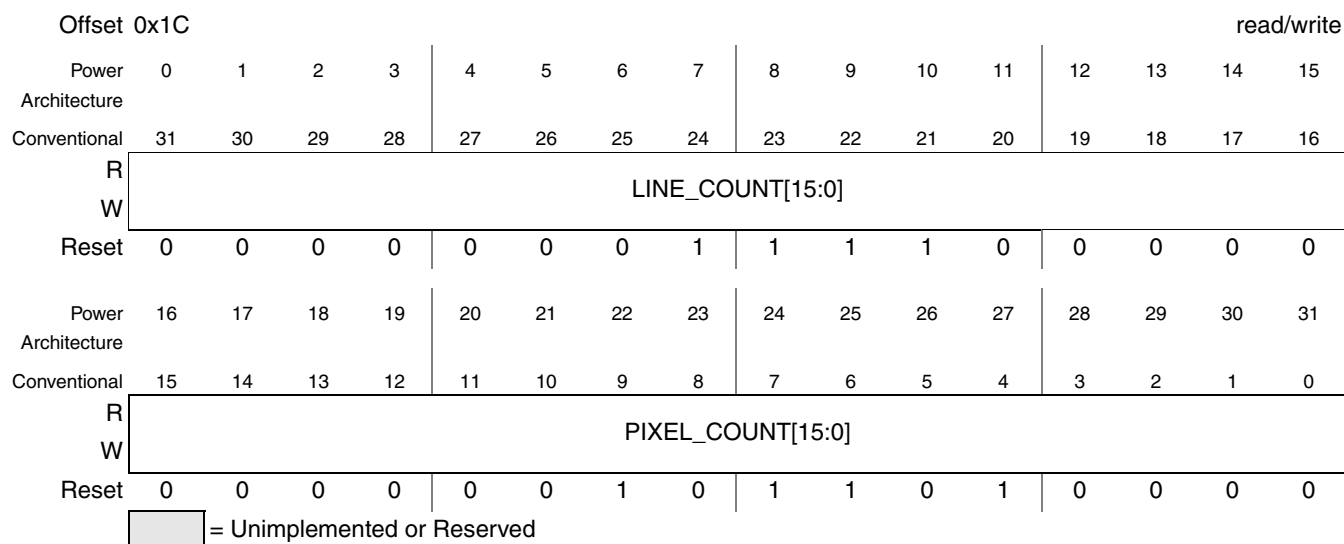


Figure 40-8. PICTURE_COUNT Register

Table 40-12. PICTURE_COUNT Fields

Field	Description
LINE_COUNT[15:0]	Expected number of active lines in each field. It is only configured when DMA is non-active, during vertical blanking. See section 40.5.2/40-17 for more details. If more lines are found during data receive part, a too many lines error interrupt is generated when ERROR_IRQ is set. Redundant lines are discarded. If less lines are found during data receive part, a not enough lines error interrupt is generated when ERROR_IRQ is set.
PIXEL_COUNT[15:0]	Expected number of active pixels in each line. It is only configured when DMA is non-active, during vertical blanking. See section 40.5.2/40-17 for more details. If more pixels are found during data receive part, a line too long error interrupt is generated when ERROR_IRQ is set. Redundant pixels are discarded. If less pixels are found during data receive part, a line too short error interrupt is generated when ERROR_IRQ is set.

40.3.3.9 HI_PRIO_ALARM

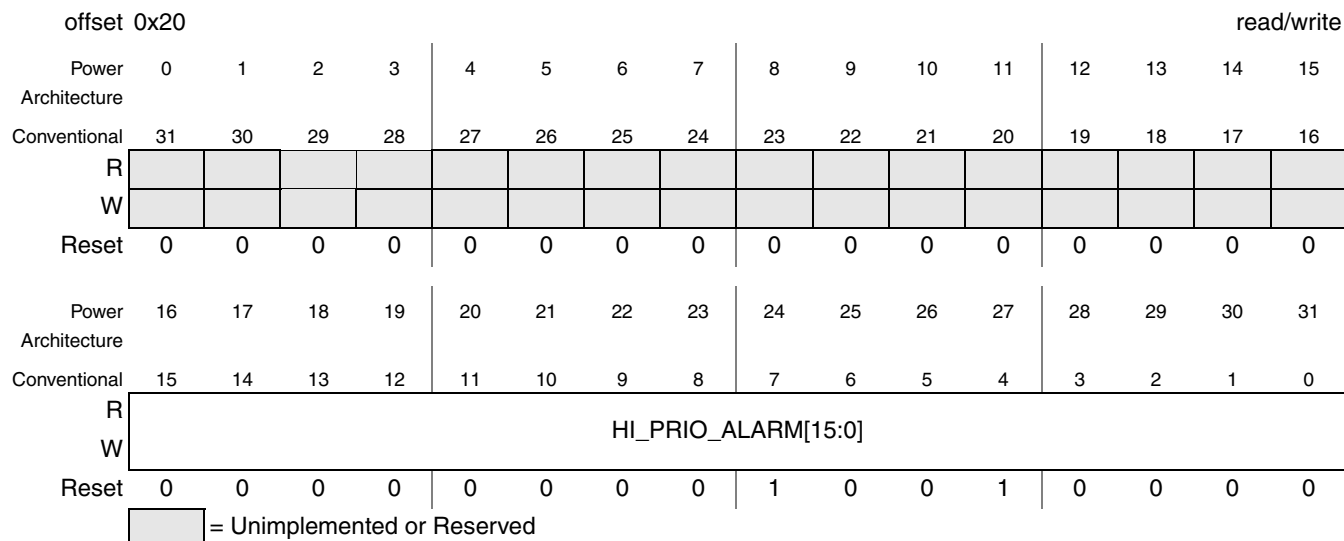


Figure 40-9. HI_PRIO_ALARM Register

Table 40-13. HI_PRIO_ALARM Fields

Field	Description
HI_PRIO_ALARM[15:0]	High priority alarm threshold. When FIFO_FILL is higher than this value, high priority bus request is asserted.

40.3.3.10 ALPHA

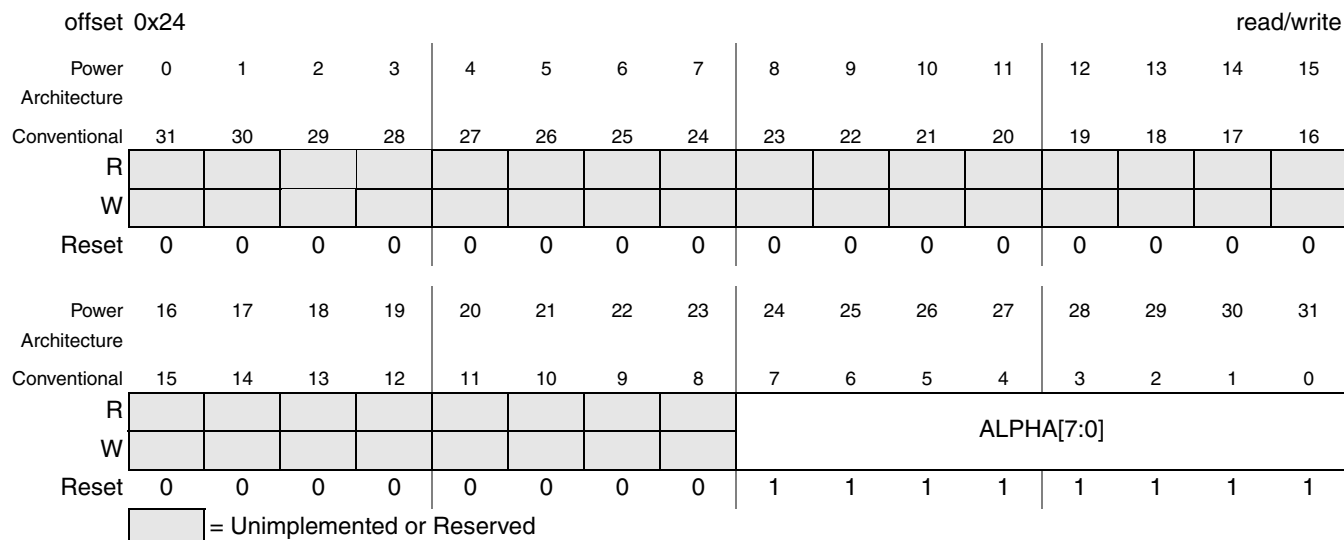


Figure 40-10. ALPHA Register

Table 40-14. ALPHA Fields

Field	Description
ALPHA[7:0]	Alpha value used for picture blending. This register is configured during vertical blanking and used from the next video field.

40.4 Functional Description

The VIDEO_IN block accepts an ITU-R BT.656-4 compatible video stream, extracts active pixel data and converts it to RGBA888/RGB565, then writes the RGB stream to memory. Ancillary data during blanking is discarded.

40.4.1 ITU656

The Recommendation ITU-R BT.656-4 describes the means of interconnecting digital television equipment operating on the 525-line or 625-line standards and complies with the 4:2:2 encoding parameters as defined in Recommendation ITU-R BT.601.

The data stream structure on ITU-R BT.656-4 interface is shown in [Figure 40-11](#). There are two timing reference signals, one at the beginning of each video data block (start of active video, SAV) and one at the end of each video data block (end of active video, EAV).

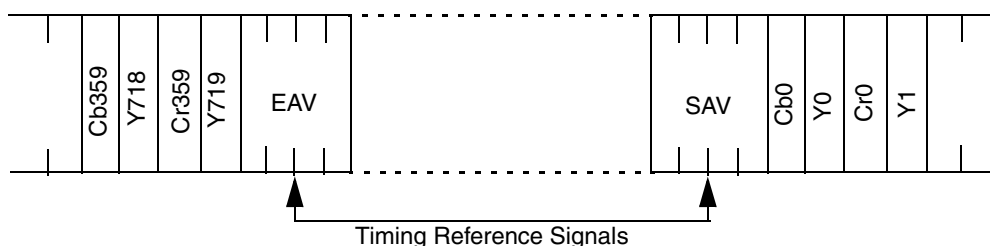


Figure 40-11. Interface Data Stream of ITU-R BT.656-4

Each timing reference signal consists of a four word sequence in the following format: FF 00 00 XY. Values are expressed in hexadecimal notation. FF 00 values are reserved for use in timing reference signals.

The first three words are a fixed preamble. The fourth word contains information defining field 2 identification, the state of field blanking, and the state of line blanking. The assignment of bits within the timing reference signal is shown in [Table 40-15](#).

Table 40-15. Video Timing Reference Codes

Data Bit Number	First Word (FF)	Second Word (00)	Third Word (00)	Fourth Word (XY)
9(MSB)	1	0	0	1
8	1	0	0	F(0: field 1, 1: field 2)
7	1	0	0	V(0: elsewhere, 1: field blanking)
6	1	0	0	H(0: in SAV, 1: in EAV)
5	1	0	0	P3

Table 40-15. Video Timing Reference Codes

Data Bit Number	First Word (FF)	Second Word (00)	Third Word (00)	Fourth Word (XY)
9(MSB)	1	0	0	1
8	1	0	0	F(0: field 1, 1: field 2)
4	1	0	0	P2
3	1	0	0	P1
2	1	0	0	P0
1	1	0	0	0
0	1	0	0	0

In above table, bits P0, P1, P2, P3 have states dependent on the states of the bits F, V and H. At the receiver this arrangement permits one-bit errors to be corrected and two-bit errors to be detected.

Refer to Recommendation ITU-R BT.656-4 for more details.

40.4.2 Round and Dither

VIDEO_IN block extracts active pixel data from ITU656 data stream depending on the timing reference signals in SAV and EAV, convert it from YCrCb to RGB888 format, and place it to an internal buffer.

Pixel data can be stored in buffer as two RGB formats: RGB888 and RGB565. When RGB888 format is selected, pixel data directly enters buffer from format converter. Otherwise, pixel data should be converted from RGB888 to RGB565. Thus the image is anamorphic more or less.

VIDEO_IN block provides two simple compensatory algorithms, round and dither, to recover the anamorphic image.

40.4.2.1 Round

When RGB888 data is converted to RGB565, the two or three LSBs are discarded and the image information carried by them is lost. To keep as much information as possible, VIDEO_IN rounds in 1 to LSB if the decimal fraction is bigger than 0.5 and ignore the smaller fraction when ROUND_ON is set in status and configuration register.

40.4.2.2 Dither

Dither is a little more complex but better than round for recovering image. It's a statistical compensation algorithm. It doesn't render all pixels with the same grey or color level, but some with the lower one, and some with a color level of 1 LSB more. The selection of adding one LSB or not depends on the position of the pixel on the screen.

Figure 41-11 shows the implementation of dither in VIDEO_IN block.

	0	0.5	0	0.5
line0	○	○	○	○
	0.25	0.75	0.25	0.75
line1	○	○	○	○
	0	0.5	0	0.5
line2	○	○	○	○
	0.25	0.75	0.25	0.75
line3	○	○	○	○

Figure 40-12. Dither Implementation

The number above pixel position in the diagram is compensation value for this pixel. When pixels have a value of 0.25, they are rendered 0 in 75% of the pixels and 1 in 25% of pixels. This averages to 0.25. Similarly, pixel values of 0.5, 0.75, and 1.0 are rendered 50%, 75% and 100% of the pixels as 1. For human eyes, this rendering result of dither makes the holistic image smoother and closer to the original one.

40.4.3 DMA and De-interlacing

VIDEO_IN block has an embedded DMA master. When video data is converted to RGB format and enters an internal buffer, it waits for conveying to memory by this internal DMA.

After doing some necessary register configuration, such as coefficients, PICTURE_COUNT and DMA_ADDRESS, user can activate DMA by setting the DMA_ACT of the status and configuration register. The DMA_ACT can be configured only during vertical blanking since VIDEO_IN block won't transfer a fragment of video field to memory. If it's configured during field active time, DMA transfer can not be started and an error interrupt is asserted.

VIDEO_IN also provides a simple way to de-interlace for interlaced video image. It is implemented by setting DMA_ADDRESS and DMA_INCREMENT registers.

Figure 40-13 shows the implementation of de-interlace. Value of DMA_INCREMENT is added to rounded address at the end of every active line. So, when DMA_INCREMENT is zero, pixel data is stored in memory line by line, meaning de-interlace is off, as shown in Figure 40-1 and Figure 40-2; otherwise, when DMA_INCREMENT equal one-line memory mapped pixel size, and $\text{DMA_ADDRESS}(2) = \text{DMA_ADDRESS}(1) + \text{DMA_INCREMENT}$, odd field and even field is merged into one frame in memory, as shown in Figure 40-3.

Here DMA_ADDRESS(1) means base address of field 1, and DMA_ADDRESS(2) means base address of field 2. Memory mapped line size means memory size that occupied by one active line pixels. It depends on pixel number of one line and video data format(RGBa888 or RGB565). See register description in section 40.3.3.7/40-10.

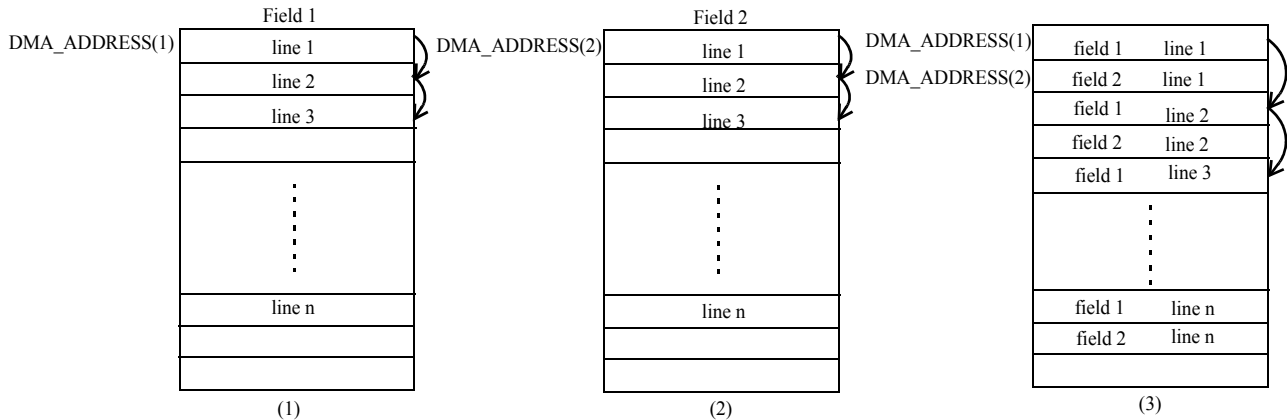


Figure 40-13. Implementation of De-interlace

40.4.4 Error Case

Normally, user should provide a standard and totally ITU-compatible video stream to VIDEO_IN block. However, it's difficult to avoid unexpected errors all the time. VIDEO_IN can manage error cases like ECC error, line too long, line too short, too many lines, or not enough line in a field, even FIFO overflow.

- ECC error: ITU stream provides 4-bit error correcting code P[3:0] in its SAV and EAV. It is decoded in VIDEO_IN to use correct field number, horizontal sync and vertical sync bits. It can correct one bit error and find two bits error. When ECC error is found, an interrupt is asserted.
- Line too long error: When pixels of active line is longer than PIXEL_COUNT, a line too long error interrupt is asserted and redundant pixels are discarded.
- Too many lines error: When active lines of a field are bigger than LINE_COUNT, a too many lines error interrupt is asserted and redundant lines is discarded.
- Line too short error: When pixels of active line is less than PIXEL_COUNT, a line too short error interrupt is asserted.
- Not enough line error: When active lines of a field is less than LINE_COUNT, a not enough line error interrupt is asserted.
- FIFO overflow error: If system bus is blocked for a long time, video data is stored in FIFO and makes FIFO overflow. When FIFO is overflow, an interrupt is asserted, and the coming data is discarded until FIFO works normally again. Current field is jumbled. However, VIDEO_IN recovers to work at the next field if bus is unblocked at that time.
- FIFO underflow: When FIFO is read when it's empty, a FIFO underflow error interrupt is asserted. Normally, this error shouldn't occur.

VIDEO_IN can manage above error cases to a certain extent. However, when it doesn't recover to work, user should write the SOFT_RESET bit of the status and configuration register to reset VIDEO_IN block.

40.5 Initialization/Application Information

40.5.1 Initialization Information

When VIDEO_IN block comes out of reset, core should implement the following steps to start this block:

1. Write VSYNC_EN and FIELD_EN bits in status register to enable field interrupt. Meanwhile, disable error interrupt.
2. When core receives vsync interrupt and field interrupt, read FIELD_NO bit of status register.
3. According to FIELD_NO bit, write DMA_ADDRESS register.
4. Configure DMA_INCREMENT and PICTURE_COUNT registers according to the video input format.
5. Clear error status first if necessary
6. Configure other required registers
7. Write DMA_ACT bit of status register to startup FIFO and DMA transfer.

40.5.2 Application Information

Normally, user should not change the register value of coefficients, PICTURE_COUNT, DMA_INCREMENT, alpha, ROUND_ON, DITHER_ON, and MODE_32BIT after VIDEO_IN is started up. When video input source is changed, user should reset VIDEO_IN and re-configure related registers.

40.5.2.1 Register Configuration Timing Window

As mentioned above, dynamic configuration on registers is not recommended because it may cause error if it's not configured in a certain timing window, especially for PICTURE_COUNT and DMA_INCREMENT.

Register configuration timing window is shown in below diagram. Here F is field identification signal, V is vertical sync signal and H is horizontal sync signal. DMA_END indicates if DMA transfer is done.

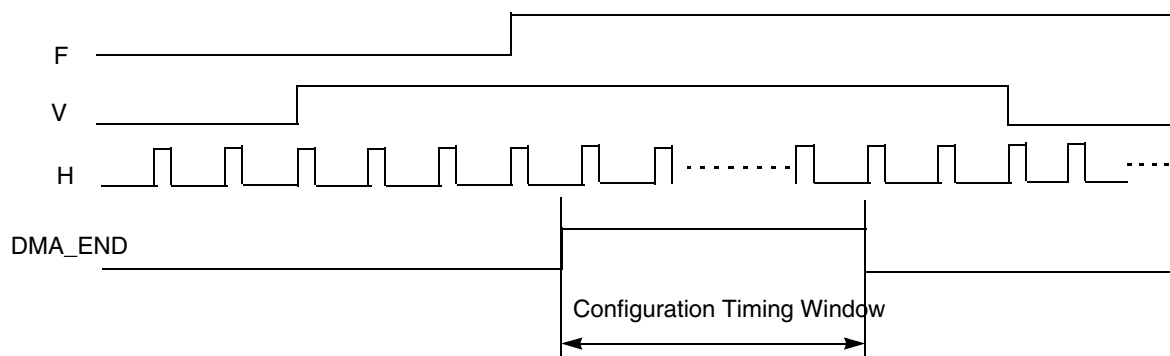
All registers, except SOFT_RESET bit, are recommended to be configured during vertical blanking, after DMA transfer is done and field identification bit is changed (field interrupt is asserted).

Normally, this window is long enough for user to complete register configuration. While in the worst case, internal buffer is full at the end of active field and it takes much time for DMA to be done, the shortest configuration window can be calculated as the following:

- RGBa888 format: $T_w = T_{\text{blanking}} - 512/F_{\text{pixel}}$
- RGB565 format: $T_w = T_{\text{blanking}} - 1024/F_{\text{pixel}}$

Here T_w is length of configuration window, and T_{blanking} is length of vertical blanking. F_{pixel} is pixel clock frequency.

For example, when pixel clock on ITU interface is 6.75MHz, T_{blanking} is 1.47ms, configuration window is $1.47\text{ms} - 1024/6.75/1000 = 1.32\text{ms}$.

**Figure 40-14. Register Configuration Timing Window**