ST52F500/F503/F504

8-BIT INTELLIGENT CONTROLLER UNIT (ICU)

Two Timer/PWMs, I^2C, SPI

TARGET SPECIFICATION

Memories

- Up to 8 Kbytes Single Voltage Flash Memory
- Up to 512 bytes of RAM
- Up to 4 Kbytes Data EEPROM
- In Situ Programming in Flash devices (ISP)
- Single byte and Page modes and In Application Programming for writing data in Flash memory
- Readout protection and flexible write protection

Core

- Register File based architecture
- 107 basic instructions
- Hardware multiplication and division
- Decision Processor for the implementation of Fuzzy Logic algorithms
- Deep System and User Stacks

Clock and Power Supply

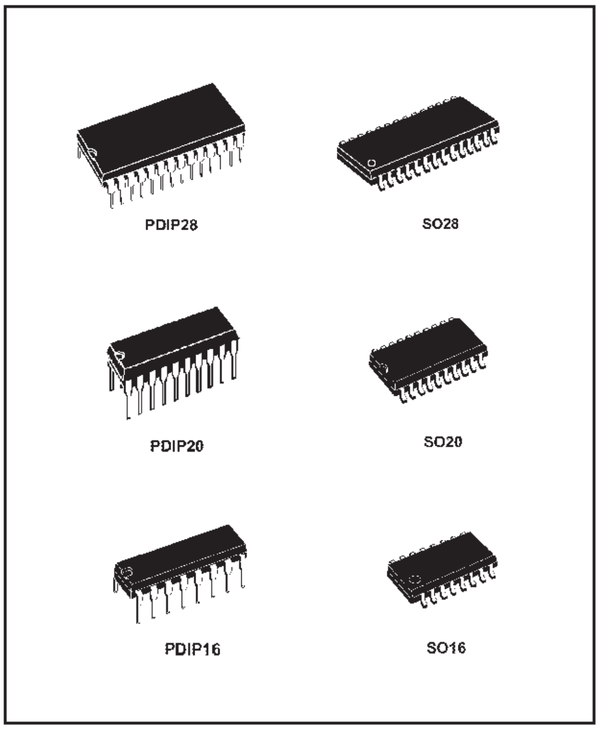
- Up to 24 MHz clock frequency
- Programmable Oscillator modes:
  - 10 MHz Internal Oscillator
  - External Clock/ Oscillator
  - External RC Oscillator
- Power-On Reset (POR)
- Programmable Low Voltage Detector (PLVD) with 3 configurable thresholds
- Power Saving features

Interrupts

- 8 interrupt vectors with one SW Trap
- Non-Maskable Interrupt (NMI)
- Two Port Interrupts with up to 16 sources

I/O Ports

- From 10 up to 22 I/O PINs configurable in pull-up, push-pull, weak pull-up, open-drain and high-impedance
- High current sink/source in all pins



Peripherals

- 2 Programmable 16 bit Timer/PWMs with internal 16-bit Prescaler featuring:
  - PWM output
  - Input capture
  - Output compare
  - Pulse generator mode
- Watchdog timer
- I^2C™ Peripheral with master and slave mode
- 3-wire SPI™ Peripheral supporting Single Master and Multi Master SPI modes

Development tools

- High level Software tools
- 'C' Compiler
- Emulator
- Low cost Programmer
- Gang Programmer

This is preliminary information on a new product foreseen to be developed. Details are subject to change without notice.

TABLE OF CONTENTS

1 GENERAL DESCRIPTION

1.1 Introduction

ST52F500/F503/F504 are devices of ST FIVE family of 8-bit Intelligent Controller Units (ICU), which can perform, both boolean and Fuzzy algorithms in an efficient manner, in order to reach the best performances that the two methodologies allow.

Produced by STMicroelectronics using the reliable high performance CMOS process for Single Voltage Flash versions, ST52F500/F503/F504 include integrated on-chip peripherals that allow maximization of system reliability, and decreased system costs in order to minimize the number of external components.

The flexible I/O configuration of ST52F500/F503/F504 allow one to interface with a wide range of external devices (for example D/A converters or power control devices), and to communicate with the most common serial standards.

ST52F500/F503/F504 pins are configurable. The user can set input or output signals on each single pin in 8 different modes, reducing the need for external components in order to supply a suitable interface with the port pins.

A hardware multiplier and divider, together with a wide instruction set, allow the implementation of complex functions by using a single instruction. Therefore, program memory utilization and computational speed is optimized.

Fuzzy Logic dedicated structures in ST52F500/F503/F504 ICU's can be exploited to model complex system with high accuracy in a useful and simple manner.

Fuzzy Expert Systems for overall system management and Fuzzy Real time Controls can be designed to increase performance at competitive costs.

The linguistic approach characterizing Fuzzy Logic is based on a set of IF-THEN rules, which describe the control behavior and on Membership Functions associated with input and output variables.

Up to 340 Membership Functions, with triangular and trapezoidal shapes, or singleton values are available to describe fuzzy variables.

The Timer/PWM peripheral allows one to manage power devices and timing signals, by implementing different operating modes and high frequency PWM (Pulse Width Modulation) controls. Input Capture and Output Compare functions are available on the Timers.

The Timer has a 16-bit programmable internal Prescaler and a 16-bit Counter, which can use internal or external START/STOP signals and clock.

An internal programmable WATCHDOG is available to avoid loop errors and reset the ICU.

ST52F500/F503/F504 supply different peripherals to implement the most common serial communication protocols. SCI allows the performance of serial asynchronous communication (UART). I^2C and SPI peripherals allow the implementation of synchronous serial protocols. I^2C peripherals can work both in master and slave mode. SPI implements Single and Multi Master modes using 3-wire.

Up to 8 interrupt vectors are available, which allow synchronization with peripherals and external devices. Non-Maskable Interrupt and S/W TRAP are available. All interrupts have configurable priority levels and are maskable excluding the Non-Maskable Interrupt, which has fixed top level priority. Two versatile Port Interrupts are available for synchronization with external sources.

The ST52F500/F503/F504 also include an on-chip Power-on-Reset (POR), which provides an internal chip reset during power up situation and a Programmable Low Voltage Detector (PLVD), which causes the ICU to reset if the voltage source V_{DD} dips below a threshold. Three programmable thresholds are available, allowing to work with different supply voltages (from 2.4 to 5.5 V).

In order to optimize energy consumption, two different power saving modes are available: Wait mode and Halt mode.

Internal Oscillator at 10 MHz $\pm$ 1% is available. External clock, quartz oscillator or RC oscillator are also applicable. The device always starts with the Internal Oscillator, then it reads an Option Byte where the clock mode to be used is programmed.

Program Memory addressing capability addresses up to 8 Kbytes of memory location to store both program instructions and data.

Memory can be locked by the user in order to prevent external undesired operations.

Operations may be performed on data stored in RAM, allowing direct combination of new inputs and feedback data. All RAM bytes are used like Register File.

An additional RAM bench is added to the Program Memory addressing space in order to allow the management of the System/User Stacks and user data storage.

ST52F500/F503/F504 supply the system stack and the user stack located in the additional RAM bench. The user stack can be located anywhere in the additional RAM by writing the top address in the configuration registers, in order to avoid overlap with other data.

Single Voltage Flash allows the user to reprogram the devices on-board by means of the In Situ

Programming (ISP) feature. It is possible to store in safe way up to 4K of data in the available EEPROM memory benches. Permanent data, both in Flash and EEPROM can be managed by means of the In-Application-Programming (IAP) feature. Single byte and Page write modes are supported. Flexible write protection, of permanent data or program instructions, is also available.

The Instruction Set composed of up to 107 instructions allows code compression and high speed in the program implementation.

A powerful development environment consisting of a board and software tools allows an easy configuration and use of ST52F500/F503/F504.

The Visual FIVE software tool allows the development and debugging of projects via a user-friendly graphical interface and optimization of generated microcode.

Third-party Hardware Emulators and ‘C’ Compiler are available to speed-up the application implementation and time-to-market.

1.2 Functional Description

ST52F500/F503/F504 ICU’s can work in two modes according to the Vpp signal levels:

- Memory Programming Mode
- Working Mode

During Working Mode Vpp must be tied to Vss. To enter the Memory Programming Mode, the Vpp pin must be tied to Vdd.

A RESET signal must be applied to the device to switch from one mode to the other.

1.2.1 Memory Programming Mode.

The ST52F500/F503/F504 memory is loaded in the Memory Programming Mode. All instructions and data are written inside the memory during this phase.

The Option Bytes are loaded during this phase by using the programming tools. The Option Bytes can only be loaded in this phase and cannot be modified run-time.

Data and commands are transmitted by using the I^2C protocol, implemented using the internal I^2C peripheral. The In-Situ Programming protocol (ISP) uses the following pins:

- SDA and SCL for transmission
- Vpp for entering in the mode
- RESET for starting the protocol in a stable status
- Vdd and Vss for the power supply.

The Internal clock is used in this phase.

1.2.2 Working Mode.

The processor starts the working phase following the instructions, which have been previously loaded in the first locations of the memory. The first instruction must be a jump to the first program instruction, skipping the data (interrupt vectors, Membership Functions, user data) stored in the first memory page.

ST52F500/F503/F504’s internal structure includes two computational blocks, the CONTROL UNIT (CU) and the DATA PROCESSING UNIT (DPU), which performs boolean functions. The DECISION PROCESSOR (DP) block cooperates with these blocks to perform Fuzzy algorithms.

The DP can manage up to 340 different Membership Functions for the antecedent part of fuzzy rules. The consequent terms of the rules are “crisp” values (real numbers). The maximum number of rules that can be defined is limited by the dimensions of the standard algorithm implemented.

The Program/Data Memory is shared between Fuzzy and standard algorithms. Within this memory, the user data can be stored both in non volatile memory as well as in the RAM locations.

The Control Unit (CU) reads information and the status of the peripherals.

Arithmetic calculus can be performed on these values by using the internal CU and Register File, which supports all computations. The peripheral inputs can be Fuzzy and/or arithmetic output values contained in the Register File or Program/ Data Memory.

Table 1.1 ST52F500/F503/F504 Devices Summary

Device	NVM	RAM	EEPROM	Timers	ADC	Comms	I/O	Package
ST52F500Ympy	1/2/4/8 K FLASH	256/512	-	2X16-bit	-	I^2C	10	Dip/So 16
ST52F500Fmpy	1/2/4/8 K FLASH	256/512	-	2X16-bit	-	I^2C SPI	14	Dip/So 20
ST52F500Gmpy	1/2/4/8 K FLASH	256/512	-	2X16-bit	-	I^2C SPI	22	Dip/So 28
ST52F503Ympy	1/2/4/8 K FLASH	256/512	128/256	2X16-bit	-	I^2C	10	Dip/So 16
ST52F503Fmpy	1/2/4/8 K FLASH	256/512	128/256	2X16-bit	-	I^2C SPI	14	Dip/So 20
ST52F503Gmpy	1/2/4/8 K FLASH	256/512	128/256	2X16-bit	-	I^2C SPI	22	Dip/So 28
ST52F504Ympy	4K FLASH	512	512/1024/ 2048/4096	2X16-bit	-	I^2C	10	Dip/So 16
ST52F504Fmpy	4K FLASH	512	512/1024/ 2048/4096	2X16-bit	-	I^2C SPI	14	Dip/So 20
ST52F504Gmpy	4K FLASH	512	512/1024/ 2048/4096	2X16-bit	-	I^2C SPI	22	Dip/So 28

Note: devices with1-2K Flash have 256 RAM / 128 EEPROM

COMMON FEATURES	ST52F500/F503/F504
Watchdog	Yes
Other Features	NMI, LVD, PLVD
Temperature Range	From -40° to +85°
Operating Supply	2.4 - 5.5 V
CPU Frequency	from 1 to 24 MHz.

Legend:

Sales code:	**ST52tnnncmpy**
Memory type (t):	F=FLASH, T=OTP, E=EPROM
Subfamily (nnn):	500, 503, 504, 510, 513, 514, 520, 521, 530
Pin Count (c):	Y=16 pins, F=20 pins, G=28 pins, K=32/34 pins, J=42/44 pins
Memory Size (m):	0=1 Kb, 1=2 Kb, 2=4 Kb, 3=8 Kb Flash (ST52x500 & ST52x503) 0=512, 1=1024, 2=2048, 3=4096 bytes EEPROM (ST52x504)
Packages (p):	B=PDIP, D=CDIP, M=PSO, T=TQFP
Temperature (y):	0=+25, 1=0 +70, 3=-40 +125, 5=-10 +85, 6=-40 +85, 7=-40 +105

Figure 1.1 ST52F500/F503/F504 Block Diagram

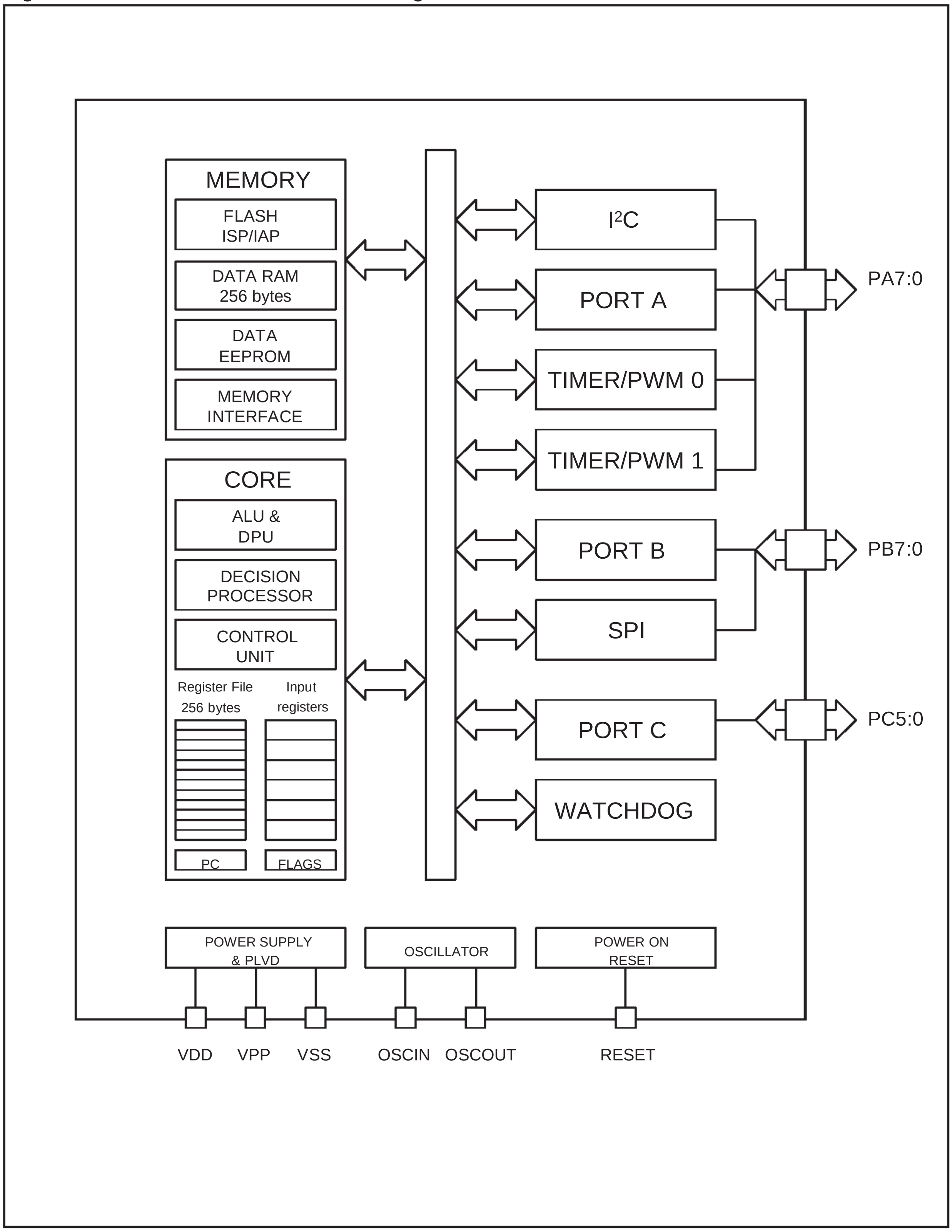


Figure 1.2 ST52F500/F503/F504 SO28/DIP28 Pin Configuration

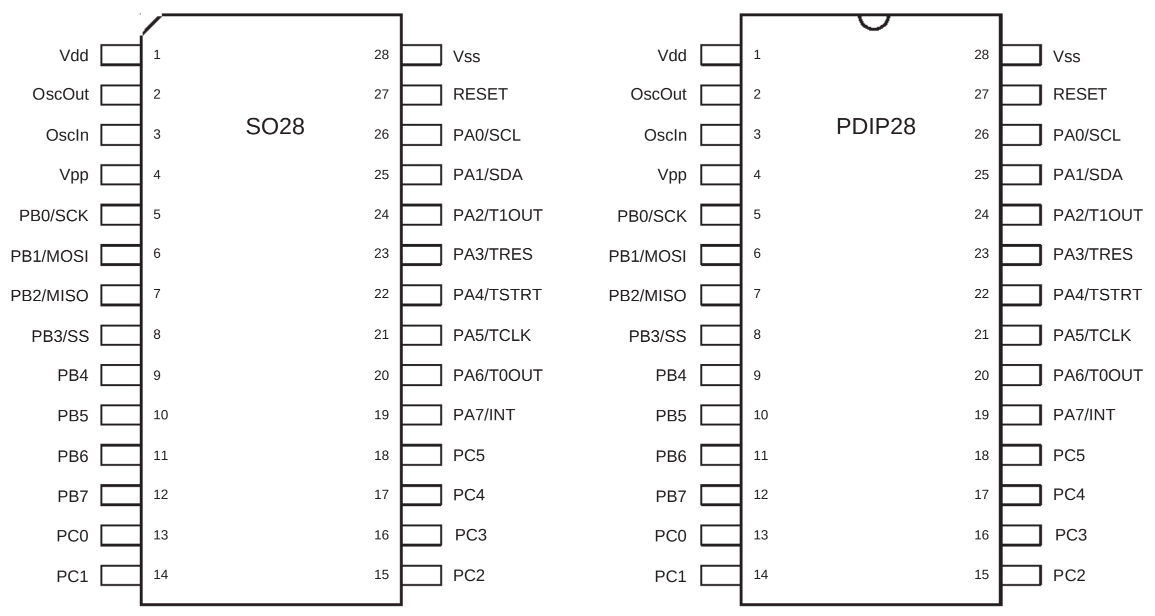


Figure 1.3 ST52F500/F503/F504 SO20/DIP20 Pin Configuration

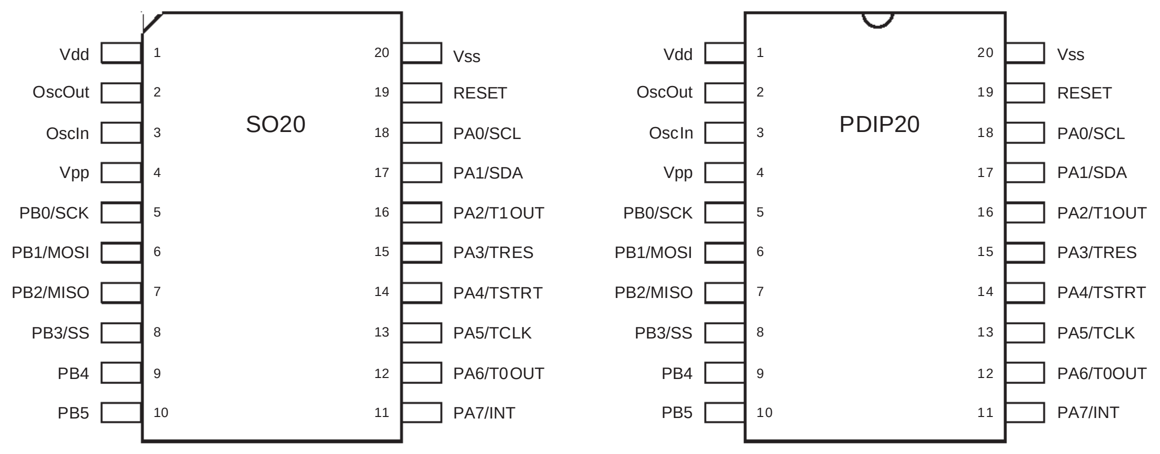


Figure 1.4 ST52F500/F503/F504 SO16/DIP16 Pin Configuration

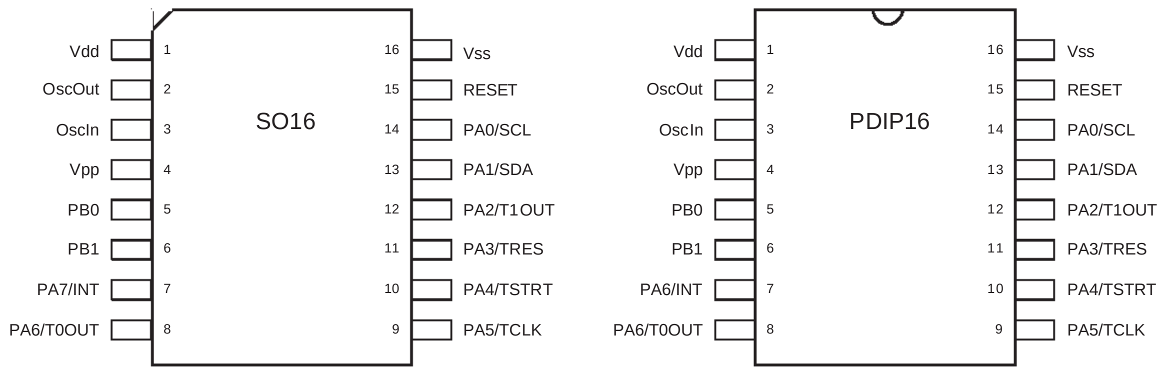


Table 1.2 ST52F500/F503/F504 Pin List

SO28 DIP28	SO20 DIP20	SO16 DIP16	NAME	Programming Phase	Working Phase
1	1	1	Vdd	Digital Power Supply	Digital Power Supply
2	2	2	OSCOUT	Oscillator Output	Oscillator Output
3	3	3	OSCIN	Oscillator Input	Oscillator Input
4	4	4	Vpp	Programming Mode Selector	Programming Mode Selector
5	5	5	PB0/SCK		SPI Serial Clock, Digital I/O
6	6	6	PB1/MOSI		SPI Master out Slave in, Digital I/O
7	7	-	PB2/MISO		SPI Master in Slave out, Digital I/O
8	8	-	PB3/SS		SPI Slave Select, Digital I/O
9	9	-	PB4		Digital I/O
10	10	-	PB5		Digital I/O
11	-	-	PB6		Digital I/O
12	-	-	PB7		Digital I/O
13	-	-	PC0		Digital I/O
14	-	-	PC1		Digital I/O
15	-	-	PC2		Digital I/O
16	-	-	PC3		Digital I/O
17	-	-	PC4		Digital I/O
18	-	-	PC5		Digital I/O
19	11	7	PA7/INT		Non Maskable Interrupt, Digital I/O
20	12	8	PA6/T0OUT		Timer/PWM 0 output, Digital I/O
21	13	9	PA5/TCLK		Timer/PWM 0 clock, Digital I/O
22	14	10	PA4/TSTRT		Timer/PWM 0 start/stop, Digital I/O
23	15	11	PA3/TRES		Timer/PWM 0 Reset, Digital I/O
24	16	12	PA2/T1OUT		Timer/PWM 1 output, Digital I/O
25	17	13	PA1/SDA	Serial Data I/O	I^2C Serial Data I/O, Digital I/O
26	18	14	PA0/SCL	Serial Clock	I^2C Serial Clock, Digital I/O
27	19	15	RESET	General Reset	General Reset
28	20	16	Vss	Digital Ground	Digital Ground

1.3 Pin Description

ST52F500/F503/F504 pins can be set in digital input mode, digital output mode, interrupt mode or in Alternate Functions. Pin configuration is achieved by means of the configuration registers. The functions of the ST52F500/F503/F504 pins are described below:

V_{DD}. Main Power Supply Voltage.

V_{SS}. Digital circuit Ground.

V_{PP}. Programming/Working mode selector. During the Programming phase V_{PP} must be set to V_{DD}. In Working phase V_{PP} must be equal to V_{SS}.

OSCin and **OSCout.** These pins are internally connected to the on-chip oscillator circuit. A quartz crystal or a ceramic resonator can be connected between these two pins in order to allow correct use of ST52F500/F503/F504 with various stability/ cost trade-offs. An external clock signal can be applied to OSCin: in this case OSCout must be grounded. To reduce costs, an RC circuit can be applied to the OSCin pin to establish the internal clock frequency, instead of the quartz. Without any connection, the device can work with its internal clock generator (10 MHz)

RESET. This signal is used to reset the ST52F500/ F503/F504 and re-initialize the registers and control signals. It is also used when switching from the Programming Mode to Working Mode and vice versa.

PA0-PA7, PB0-PB7,PC0-PC5. These lines are organized as I/O ports. Each pin can be configured as an input, output (with pull-up, push-pull, weak-pull-up, open-drain, high-impedance), or as an interrupt source.

T0OUT, T1OUT. These pins output the signals generated by the PWM/Timer 0 and PWM/Timer 1 peripheral.

TRES, **TSTRT, TCLK** . These pins are related to the PWM/Timer 0 peripheral and are used for Input Capture and event counting. The TRES pin is used to set/reset the Timer; the TSTRT pin is used to start/stop the counter. The Timer can be driven by the internal clock or by an external signal connected to the TCLK pin.

INT. This pin is used as input for the Non-Maskable (top level) interrupt. The interrupt signal is detected only if the pin is configured in Alternate Function.

SCL, SDA. These pin are used respectively as Serial Clock and Serial Data I/O in I^2C peripheral protocol. They are used also in Programming Mode to receive and transmit data.

SCK, MISO, MOSI, SS. These pins are used by the Serial Peripheral Interface (SPI) peripheral. SCK is the serial clock line. MISO (Master In Slave Out) and MOSI (Master Out Slave In) are the serial data lines, which work in input or in output depending on if the device is working in slave or master mode. The SS pin allows the selection of the device master/slave mode.

2 INTERNAL ARCHITECTURE

ST52F500/F503/F504's architecture is Register File based and is composed of the following blocks and peripherals:

- Control Unit (CU)
- Data Processing Unit (DPU)
- Decision Processor (DP)
- ALU
- Memory Interface
- up to 256 bytes Register File
- Program/Data Memory
- Data EEPROM
- Interrupts Controller
- Clock Oscillator
- PLVD and POR
- Digital I/O ports
- Timer/PWMs
- I^2C
- SPI

2.1 Control Unit and Data Processing Unit

The Control Unit (CU) decodes the instructions stored in the Program Memory and generates the appropriate control signals. The main parts of the CU are illustrated in Figure 2.1.

The five different parts of the CU manage Loading, Logic/Arithmetic, Jump, Control and the Fuzzy instruction set.

The block called "Collector" manages the signals deriving from the different parts of the CU. The collector defines the signals for the Data Processing Unit (DPU) and Decision Processor (DP), as well as for the different peripherals of the ICU.

The block called "Arbiter" manages the different parts of the CU, so that only one part of the system is activated during working mode.

The CU structure is extremely flexible and was designed with the purpose of easily adapting the core of the microcontroller to market needs. New instruction sets or new peripherals can easily be included without changing the structure of the microcontroller, maintaining code compatibility.

A set of 107 different instructions is available. Each instruction requires a number of clock pulses to be performed that depends on the complexity of the instruction itself. The clock pulses to execute the instructions are driven directly by the masterclock, which has the same frequency of the oscillator signal supplied.

Figure 2.1 CU Block Diagram

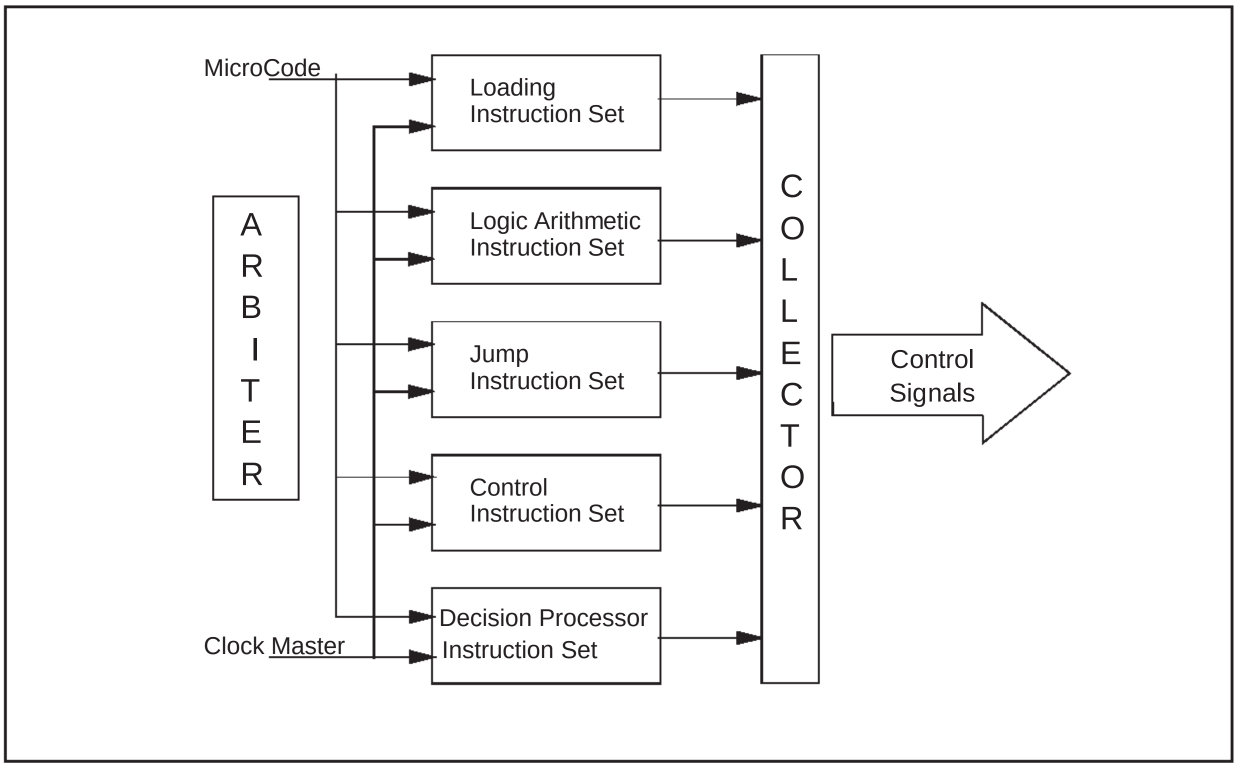
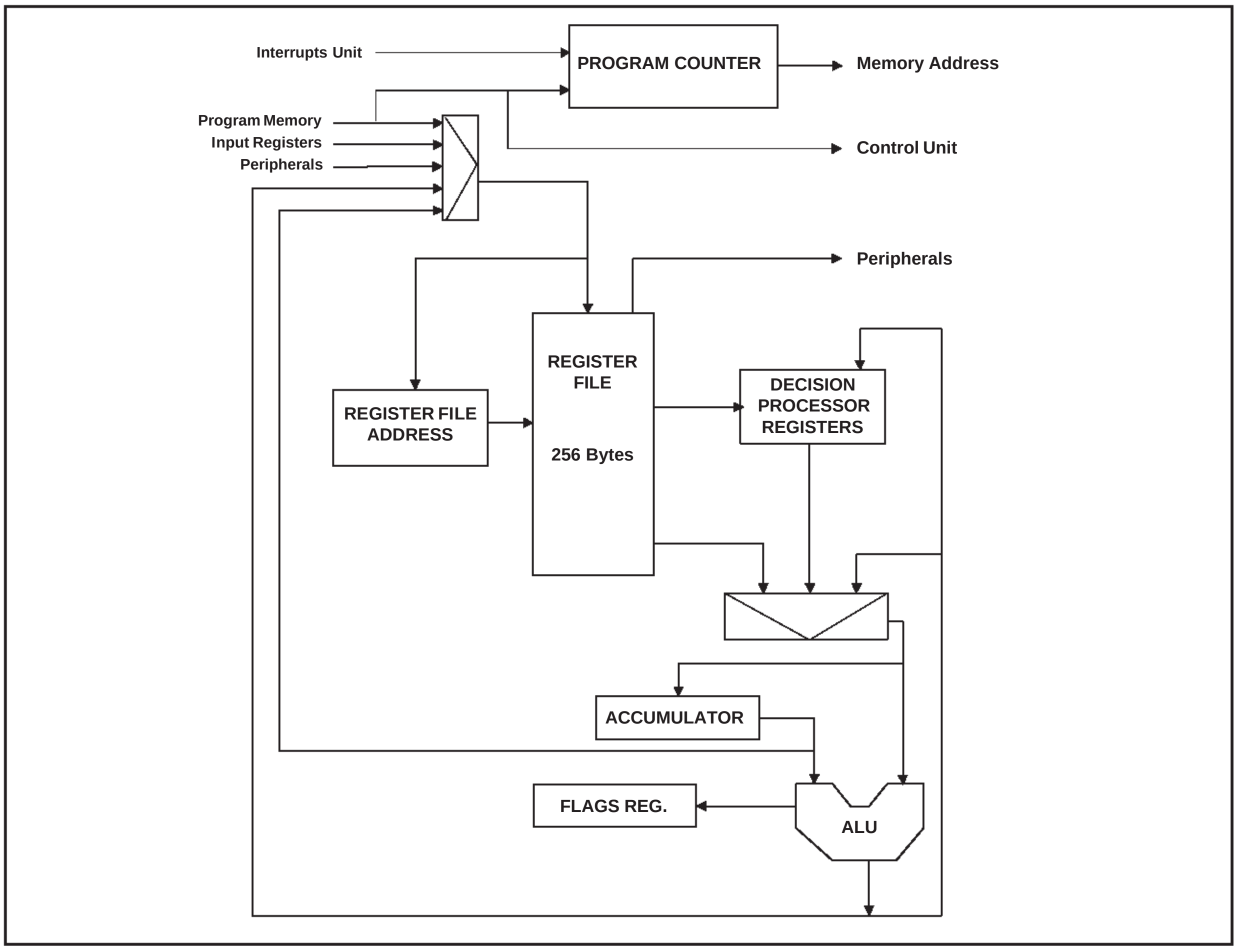


Figure 2.2 Data Processing Unit (DPU)



The DPU receives, stores and sends the instructions deriving from the Program/Data Memory, Register File or from the peripherals. It is controlled by the CU on the basis of the decoded instruction. The Fuzzy registers store the partial results of the fuzzy computation. The accumulator register is used by the ALU and is not accessible directly: the instructions used by the ALU can address all the Register File locations as operands, allowing a more compact code and a faster execution.

The following addressing modes are available: inherent, immediate, direct, indirect, bit direct.

2.1.1 Program Counter.

The Program Counter (PC) is a 16-bit register that contains the address of the next memory location to be processed by the core. This memory location may be both an instruction or data address.

The Program Counter's 16-bit length allows the direct addressing of a maximum of 64 Kbytes in the Program/Data Memory space.

The PC can be changed in the following ways:

■ JP (Jump)	PC = Jump Address
■ Interrupt	PC = Interrupt Vector
■ RETI	PC = Pop (stack)
■ RET	PC = Pop (stack)
■ CALL	PC = Subroutines address
■ Reset	PC = Reset Vector
■ Normal Instruction	PC = PC + 1

2.1.2 Flags.

The ST FIVE core includes different sets of flags that correspond to 2 different modes: normal mode and interrupt mode. Each set of flags consist of a CARRY flag (C), ZERO flag (Z) and SIGN flag (S). Each set is stacked: one set of flags is used during normal operation and other sets are used during each level of interrupt. Formally, the user has to manage only one set of flags: C, Z and S since the flag stack operation is performed automatically.

Each interrupt level has its own set of flags, which is saved in the Flag Stack during interrupt servicing. These flags are restored from the Flag Stack automatically when a RETI instruction is executed.

If the ICU was in normal mode before an interrupt, after the RETI instruction is executed, the normal flags are restored.

Note: *A subroutine CALL is a normal mode execution. For this reason a RET instruction, consequent to a CALL instruction, doesn't affect the normal mode set of flags.*

Flags are not cleared during context switching and remain in the state they were in at the exit of the last interrupt routine switching.

The Carry flag is set when an overflow occurs during arithmetic operations, otherwise it is cleared.

The Sign flag is set when an underflow occurs during arithmetic operations, otherwise it is cleared.

2.2 Arithmetic Logic Unit

The 8-bit Arithmetic Logic Unit (ALU) performs arithmetic calculations and logic instructions such as: sum, subtraction, bitwise AND, OR, XOR, bit set and reset, bit test and branch, right/left shift and rotate (see the Chapter 9 Instruction Set for further details).

In addition, the ALU of ST52x520/521 can perform multiplication (MULT) and division (DIV). Multiplication is performed by using 8 bit operands storing the result in 2 registers (16 bit values); the division instruction addresses the MSB of the dividend (the LSB is stored in the next address): the result and remainder are stored in these source addresses (see Figure 2.3 and Figure 2.4).

In order to manage signed type values, the ALU also performs addition and subtraction with offset (ADDO and SUBO). These instructions respectively subtract and add 128 to the overall result, in order to manage values logically in the range between -128,127.

Figure 2.3 Multiplication

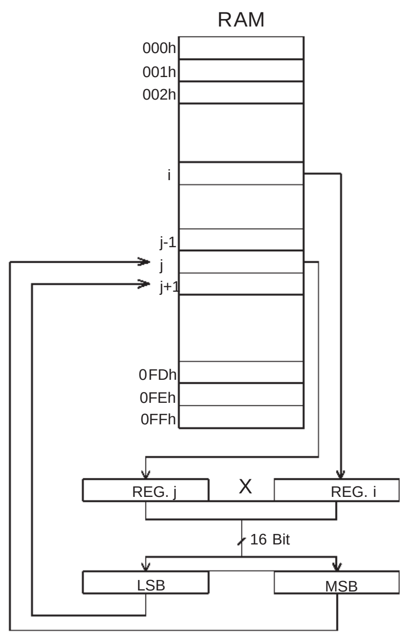
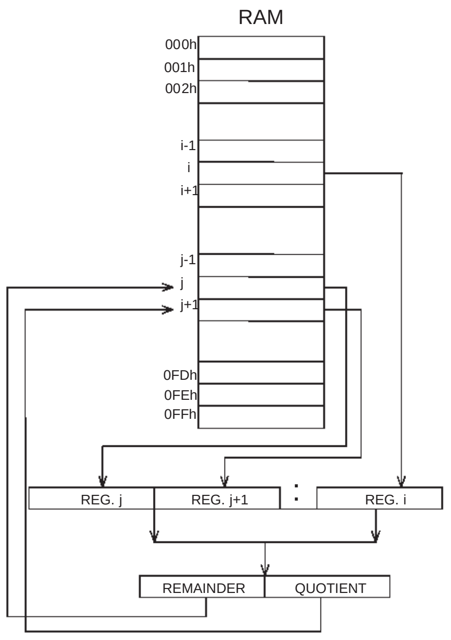


Figure 2.4 Division



3 ADDRESSING SPACES

ST52F500/F503/F504 has six separate addressing spaces:

- Register File
- Program/Data Memory
- Stacks
- Input Registers
- Output Registers
- Configuration Registers

Each space is addressed by a load type instruction that indicates the source and the destination space in the mnemonic code (see Figure 3.1).

3.1 Memory Interface

The read/write operation in the space addresses are managed by the Memory Interface, which can recognize the type of memory addressed and set the appropriate access time and mode.

In addition, the Memory Interface manages the In Application Programming (IAP) functions in Flash devices like writing cycle and memory write protection.

3.2 Register File

The Register File consists of up to 256 general purpose 8-bit RAM locations called “registers” in order to recall the functionality.

The Register File exchanges data with all the other addressing spaces and is used by the ALU to perform all the arithmetic and logic instructions. These instructions have any Register File address as operands.

Data can be moved from one location to another by using the LDRR instruction; see further ahead for information on the instruction used to move data between the Register File and the other addressing spaces.

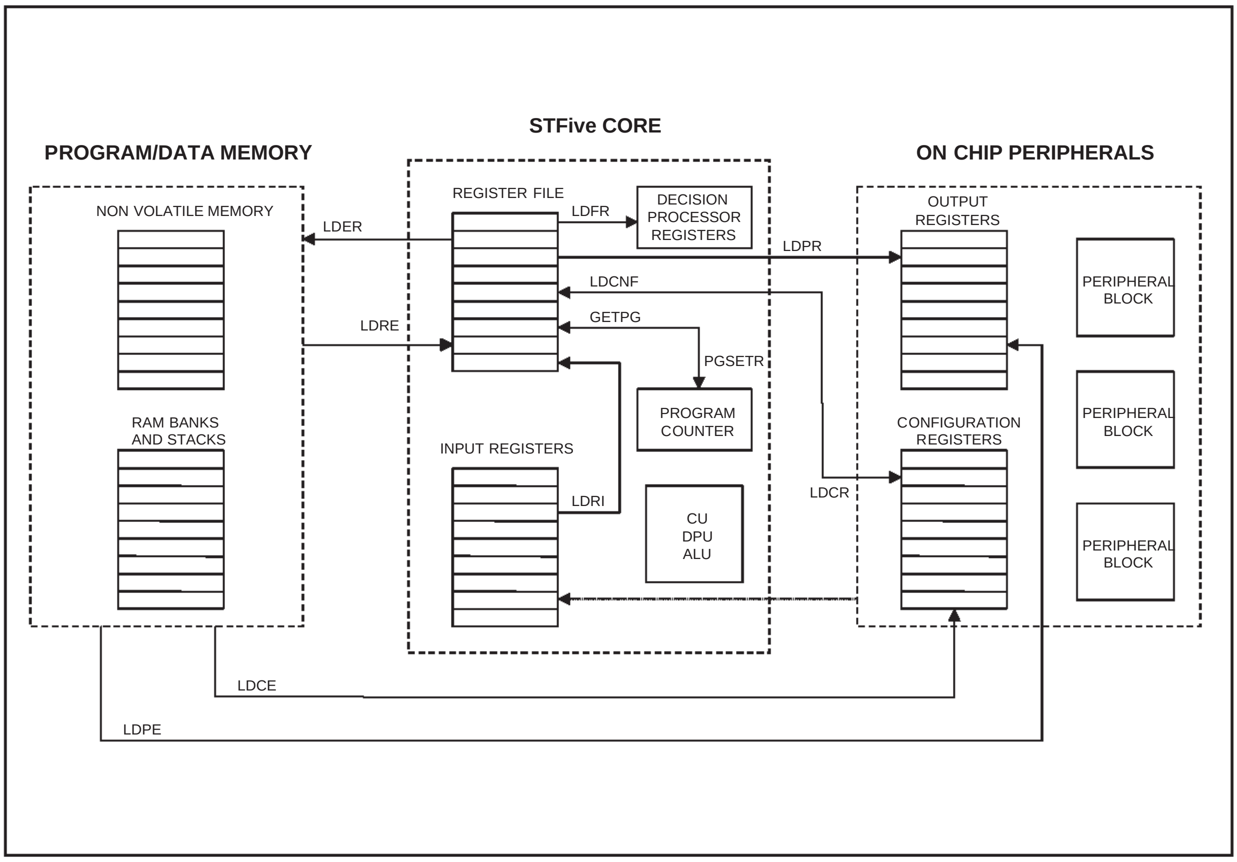
3.3 Program/Data Memory

The Program/Data Memory consists of both non-volatile memory (Flash, EEPROM) and RAM memory benches.

Non-volatile memory (NVM) is mainly used to store the user program and can also be used to store permanent data (constant, look-up tables).

Each RAM bench consists of 128/256 locations used to store run-time user data. At least one bench is present in the devices. RAM benches are also used to implement both System and User Stacks.

Figure 3.1 Addressing Spaces

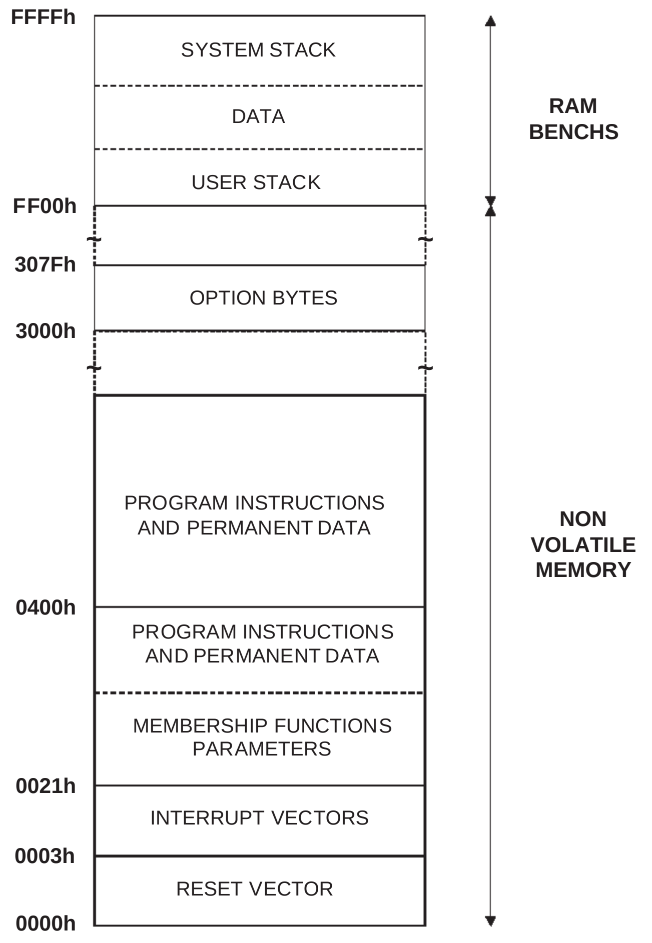


NVM is always located beginning after the first locations of the addressing space. RAM banks are always located after NVM.

NVM is organized in accordance to the following blocks (see Figure 3.2):

- *Reset Vector block* (from address 0 to 2) contains an absolute jump instruction to the first user program instruction. The Assembler tool automatically fills these locations with correct data.
- *Interrupt Vectors block* (from location 3 up to 32) contains the interrupt vectors. Each address is composed of three bytes (the jump opcode and the 16 bit address). Interrupt vectors are set by using IRQ pseudo-instruction (see the Programming Manual).
- *Mbfs Setting block* (just after the interrupt vectors) contains the coordinates of the vertexes of every Mbf defined in the program. The last address that can be assigned to this block is 1023. This area is dynamically assigned according to the size of the fuzzy routines. The memory area that remains unused, if any, is assigned to the Program Instructions block.
- *The Program Instructions block* (just after the last Mbf data through the last NVM address) contains the instruction of the user program and the permanent data.
- *Option bytes block* (from location 3000h to 307Fh) is the addressing space reserved for the option bytes. In ST52F500/F503/F504, only the location from 3000h to 3007h are used.

Figure 3.2 Program/Data Memory Organization



Flash and EEPROM are programmed electrically just applying the supply voltage (2.4 V to 5.5 V) and it is also erased electrically; this feature allows the user to easily reprogram the memory without taking the device off from the board (In Situ Programming ISP). Data and commands are transmitted through the I^2C serial communication protocol. Data can also be written run-time with the In Application Programming (IAP)

NVM can be locked by the user during the programming phase, in order to prevent external operation such as reading the program code and assuring protection of user intellectual property. Flash and EEPROM pages can be protected by unintentional writings.

The operations that can be performed on the NVM during the Programming Phase, ISP and IAP are described in detail in the Section 4.

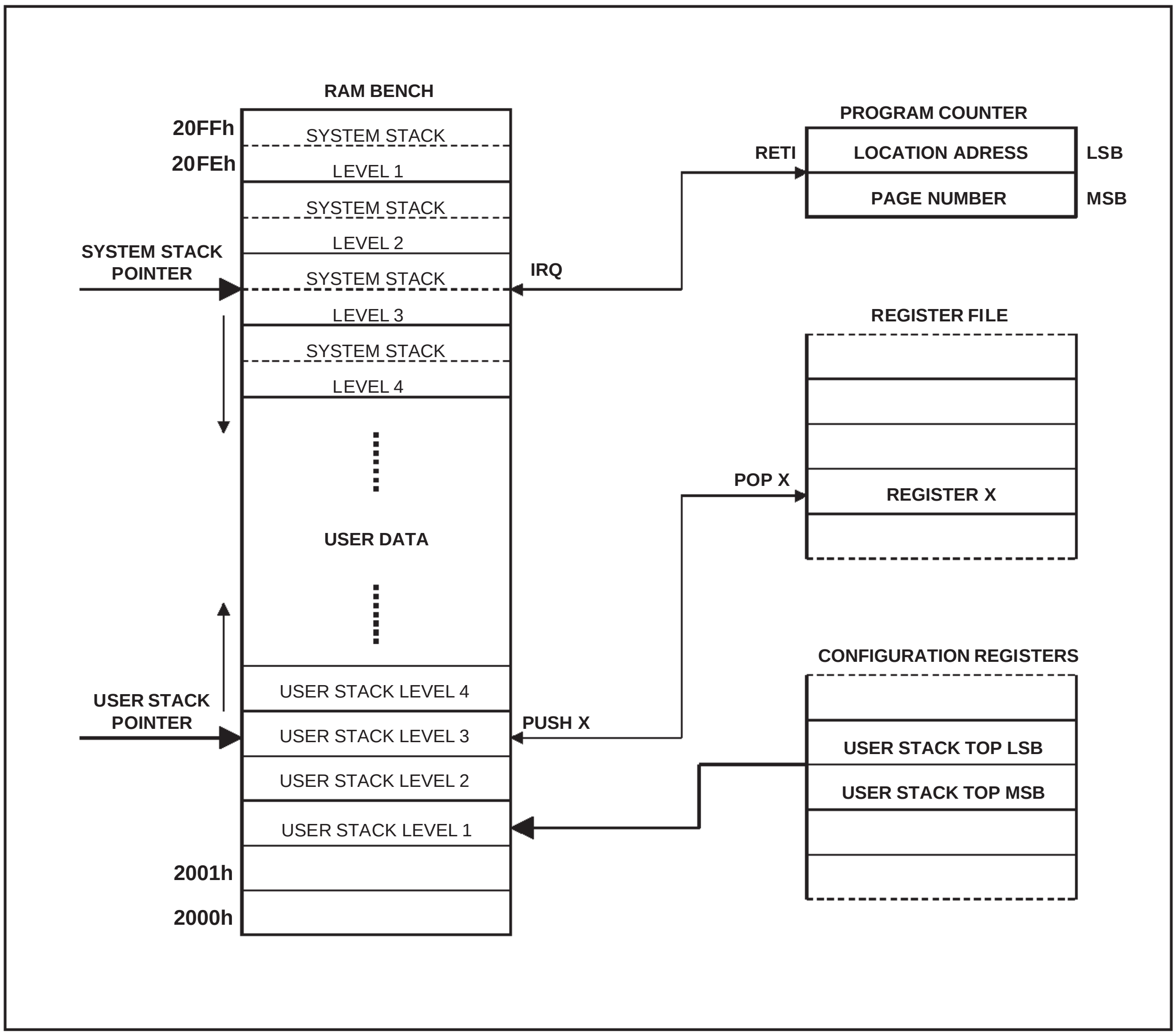
3.4 System and User Stacks

The System and User Stacks are located in the Program/Data memory in the RAM benches.

System Stacks are used to push the Program Counter (PC) after an Interrupt Request or a Subroutine Call. After a RET (Return from a subroutine) or a RETI (Return from an interrupt) the PC that is saved is popped from the stack and restored. After an interrupt request, the flags are also saved in a reserved stack inside the core, so each interrupt has its own flags.

The System Stack is located in the last RAM bench starting from the last address (255) inside the bench page. The System Stack Pointer (SSP) can be read and modified by the user. For each level of stack 2 bytes of the RAM are used. The SSP points to the first currently available stack position. When a subroutine call or interrupt request occurs, the content of the PC is stored in a couple of locations pointed to by the SSP that is decreased by 2.

Figure 3.3 System and User Stack



When a return occurs (RET or RETI instruction), the SSP is increased by 2 and the data stored in the pointed locations couple is restored back into the PC.

The current SSP can be read and write in the couple of Configuration Registers 44 02Ch (MSB: page number, always 32 020h) and 45 02Dh (LSB: location address) (see Figure 3.3). In ST52F500/F503/F504 the user can only consider the LSB because the MSB is always the same.

The User Stack is used to store user data and is located beginning from a RAM bench location set by the user (USTP) by writing the couple of Configuration Registers 5 005h (MSB: page number) and 6 005h (LSB: location address) (see Figure 3.3). Register 5, which is the page number, must always be set to a value between 32 (020h) and 255 (0FFh): values higher than 32 always address RAM on page 32.

This feature allows a flexible use of the User Stack in terms of dimension and to avoid overlaps. The User Stack Pointer (USP) points to the first currently available stack location. When the user stores a byte value contained in the Register File by using the PUSH instruction, the value is stored in the position pointed to by the USP that is increased (the User Stack order is opposite to the System Stack one). When the user takes a value from the User Stack with the POP instruction, the USP is decreased and the value pointed to is copied in the specified Register File location.

By writing the USTP, the new address is automatically written in the USP. The current USP can be read from the Input Registers 75 04Bh (MSB: page number, always 32 020h) and 76 04Ch (LSB: location address) (see Figure 3.3). In ST52F500/F503/F504 the user can only consider the LSB because the MSB is always the same.

Note: *The user must pay close attention to avoid overlapping user and Stacks data. The User Stack Top location and the System Stack Pointer should be configured with care in order to have enough space between the two stacks.*

3.5 Input Registers

The ST52F500/F503/F504 Input Registers bench consists of a file of 8-bit registers containing data or the status of the peripherals. For example, the Input Registers contain data converted by the ADC, Ports, serial communication peripherals, Timers, etc.

The Input Registers can be accessed by using the LDRI instruction that loads the specified Register File address with the contents of the specified Input Register. See the Programming Manual for further details on this instruction. The Input Registers are read-only registers.

In order to simplify the concept, a mnemonic name is assigned to each register. The same name is used in Visual FIVE development tools. The list of the Input Registers is shown in Table 3.1.

3.6 Output registers

The ST52F500/F503/F504 Output Registers bench consists of a file of 8-bit registers containing data sent to the Peripherals and the I/O Ports (for example: Timer Counters, data to be transmitted by the serial communication peripherals, data to be sent to the Port pins in output, etc.).

The registers are located inside the Peripherals and Ports, which allow flexibility and modularity in the design of new family devices.

The Output Registers are write only. In order to access the configuration Register the user can use the following instructions:

- LDPI: loads the immediate value in the specified Output Register.
- LDPR: loads the contents of the specified Register File location into the output register specified. This instruction allows computed data to be sent to Peripherals and Ports.
- LDPE direct: loads the contents of the specified Program/Data Memory location into the output register specified. This instruction allows data to be sent to Peripherals and Ports from a table.
- LDPE indirect: loads the contents of the Program/Data Memory location whose address is contained in the specified Register File location into the output register specified. This instruction allows data to be sent to Peripherals and Ports from a table pointed to by a register.

See the Programming manual for further details about these instructions.

In order to simplify the concept, a mnemonic name is assigned to each register. The same name is used in Visual FIVE development tools. The list of the Output Registers is shown in Table 3.2.

3.7 Configuration Registers & Option Bytes

The ST52F500/F503/F504 Configuration Registers bench consists of a file of 8-bit registers that allows the configuration of all the ICU blocks. The registers are located inside the block they configure in order to obtain greater flexibility and modularity in the design of new family devices. In the Configuration Registers, each bit has a peculiar use, so the logic level of each of them must be considered.

Some special configuration data, that needs to be load at the start-up and not further changed, are stored in Option Bytes. These are loaded only during the device programming phase. See Table 3.3 and Section 4 for a detailed description of the Option Bytes.

The Configuration Registers are readable and writable; the addresses refer to the same register both in read and in write. In order to access the Configuration Register the user can work in several modes by utilizing the following instructions:

- LDCI: loads the immediate value in the Configuration Register specified and is the most commonly used to write configuration data.
- LDCR: loads the Configuration Register specified with the contents of the specified Register File location, allowing a parametric configuration.
- LDCE: loads the Configuration Register specified with the contents of the specified Program/Data Memory location, allowing the configuration data to be taken from a table.
- LDCNF: loads the Register File location specified with the contents of the Configuration Register indicated, allowing for the inspection of the configuration of the device (permitting safe run-time modifications).

In order to simplify the concept, a mnemonic name is assigned to each register. The same name is used in Visual FIVE development tools. The list of the Configuration Registers is shown in Table 3.4.

Table 3.1 Input Registers

Mnemonic	Description	Address	
PORT_A_IN	Port A data Input Register	0	00h
PORT_B_IN	Port B data Input Register	1	01h
PORT_C_IN	Port C data Input Register	2	02h
-	Not Used	3	03h
-	Not Used	4	04h
SPI_IN	Serial Peripheral Interface data Input Register	5	05h
I2C_IN	I^2C Interface data Input Register	6	06h
I2C_SR1	I^2C Interface Status Register 1	7	07h
I2C_SR2	I^2C Interface Status Register 2	8	08h
-	Not Used	9	09h
-	Not Used	10	0Ah
USP_H	User Stack Pointer (MSB)	11	0Bh
USP_L	User Stack Pointer (LSB)	12	0Ch
CHAN0	A/D Channel 0 converted data	13	0Dh
-	Not Used	14-20	0Eh-014h
PWM0_COUNT_IN_H	PWM/Timer 0 Counter Input Register (MSB)	21	015h
PWM0_COUNT_IN_L	PWM/Timer 0 Counter Input Register (LSB)	22	016h
PWM0_STATUS	PWM/Timer 0 Status Register	23	017h
PWM0_CAPTURE_H	PWM/Timer 0 Capture Register (MSB)	24	018h
PWM0_CAPTURE_L	PWM/Timer 0 Capture Register (LSB)	25	019h
PWM1_COUNT_IN_H	PWM/Timer 1 Counter Input Register (MSB)	26	01Ah
PWM1_COUNT_IN_L	PWM/Timer 1 Counter Input Register (LSB)	27	01Bh
PWM1_STATUS	PWM/Timer 1 Status Register	28	01Ch
PWM1_CAPTURE_H	PWM/Timer 1 Capture Register (MSB)	29	01Dh
PWM1_CAPTURE_L	PWM/Timer 1 Capture Register (LSB)	30	01Eh
-	Not Used	31-37	01Fh-025h
FLAGS	Flag Register	38	026h
-	Not Used	39	027h
IAP_SR	In Application Programming Status Register	40	028h

Table 3.2 Output Registers

Mnemonic	Description	Address	
PORT_A_OUT	Port A data Output Register	0	00h
PORT_B_OUT	Port B data Output Register	1	01h
PORT_C_OUT	Port C data Output Register	2	02h
-	Not Used	3	03h
-	Not Used	4	04h
SPI_OUT	Serial Peripheral Interface data Output Register	5	05h
I2C_OUT	I^2C Interface data Output Register	6	06h
PWM0_COUNT_OUT_H	PWM/Timer 0 Counter Output Register (MSB)	7	07h
PWM0_COUNT_OUT_L	PWM/Timer 0 Counter Output Register (LSB)	8	08h
PWM0_RELOAD_H	PWM/Timer 0 Reload Register (MSB)	9	09h
PWM0_RELOAD_L	PWM/Timer 0 Reload Register (LSB)	10	0Ah
PWM1_COUNT_OUT_H	PWM/Timer 1 Counter Output Register (MSB)	11	0Bh
PWM1_COUNT_OUT_L	PWM/Timer 1 Counter Output Register (LSB)	12	0Ch
PWM1_RELOAD_H	PWM/Timer 1 Reload Register (MSB)	13	0Dh
PWM1_RELOAD_L	PWM/Timer 1 Reload Register (LSB)	14	0Eh

Table 3.3 Option Bytes

Mnemonic	Description	Address	
OSC_CR	Oscillator Control Register	0	00h
CLK_SET	Clock Parameters	1	01h
OSC_SET	Oscillator Set-Up	2	02h
PLDV_CR	Programmable Low Voltage Detector Control Register	3	03h
WDT_EN	HW/SW Watchdog selector	4	04h
PG_LOCK	First Page Write Protected	5	05h
PG_UNLOCK	First Page not Write Protected	6	06h
WAKEUP	Wake Up from Halt Time	7	07h

Table 3.4 Configuration Registers

Mnemonic	Description	Address	
INT_MASK	Interrupt Mask Register	0	00h
INT_POL	Interrupts Polarity and LVD enable Register	1	01h
INT_PRL_H	Interrupt Priority Register (higher priority)	2	02h
INT_PRL_M	Interrupt Priority Register (medium priority)	3	03h
INT_PRL_L	Interrupt Priority Register (lower priority)	4	04h
USTP_H	User Stack Top Pointer (MSB)	5	05h
USTP_L	User Stack Top Pointer (LSB)	6	06h
WDT_CR	Watchdog Configuration Register	7	07h
-	not used	8	08h
PWM0_CR1	PWM/Timer 0 Configuration Register	9	09h
PWM0_CR2	PWM/Timer 0 Configuration Register	10	0Ah
PWM0_CR3	PWM/Timer 0 Configuration Register	11	0Bh
PWM1_CR1	PWM/Timer 1 Configuration Register	12	0Ch
PWM1_CR2	PWM/Timer 1 Configuration Register	13	0Dh
-	Not Used	14	0Eh
-	Not Used	15	0Fh
I2C_CR	I^{2}C Interface Control Register	16	010h
I2C_CCR	I^{2}C Interface Clock Control Register	17	011h
I2C_OAR1	I^{2}C Interface Own Address Register 1	18	012h
I2C_OAR2	I^{2}C Interface Own Address Register 2	19	013h
SPI_CR	Serial Peripheral Interface Control Register	20	014h
SPI_STATUS_CR	Serial Peripheral Interface Control-Status Register	21	015h
-	Not Used	22	016h
-	Not Used	23	017h
PORT_A_PULLUP	Port A Pull Up enable/disable Register	24	018h
PORT_A_OR	Port A Option Register	25	019h

Table 3.4 Configuration Registers

Mnemonic	Description	Address	
PORT_A_DDR	Port A Data Direction Register	26	01Ah
PORT_A_AF	Port A Alternate Function selection Register	27	01Bh
PORT_B_PULLUP	Port B Pull Up enable/disable Register	28	01Ch
PORT_B_OR	Port B Option Register	29	01Dh
PORT_B_DDR	Port B Data Direction Register	30	01Eh
PORT_B_AF	Port B Alternate Function selection Register	31	01Fh
PORT_C_PULLUP	Port C Pull Up enable/disable Register	32	020h
PORT_C_OR	Port C Option Register	33	021h
PORT_C_DDR	Port C Data Direction Register	34	022h
-	Not Used	35-43	023h-02Bh
SSP_H	System Stack Pointer (MSB)	44	02Ch
SSP_L	System Stack Pointer (LSB)	45	02Dh
CPU_CLK	CPU Clock Prescaler	46	02Eh

4 MEMORY PROGRAMMING

ST52F500/F503/F504 provides an on-chip user programmable non-volatile memory, which allows fast and reliable storage of user data.

Program/Data Memory addressing space is composed by a Single Voltage Flash Memory and a RAM memory bench. The ST52F503/504 devices also have a Data EEPROM bench to store permanent data with long term retention and a high number of write/erase cycles.

All the Program Data memory addresses can execute code, including RAM and EEPROM benches.

The memory is programmed by setting the V_{pp} pin equal to V_{dd}. Data and commands are transmitted through the I^2C serial communication protocol. The same procedure is used to perform "In-Situ" the programming of the device after it is mounted in the user system. Data can also be written in run-time with the In-Application Programming (IAP).

The Memory can be locked by the user during the programming phase, in order to prevent external operation such as reading the program code and assuring protection of user intellectual property. Flash and EEPROM pages can be protected by unintentional writings.

4.1 Program/Data Memory Organization

The Program/Data Memory is organized as described in Section 3.3. The various sales types have different amounts of each type of memory. Table 4.1 describes the memory benches amount and page allocation for each sales type.

***Remark:** some devices have RAM or EEPROM memory benches of 128 bytes. The address range inside the page of these benches is between 128 to 255.*

The addressing spaces are organized in pages of 256 bytes. Each page is composed by blocks of 32 bytes. Memory programming is performed one block at a time in order to speed-up the programming time (about 2.5 ms per block).

The whole location address is composed as follows:

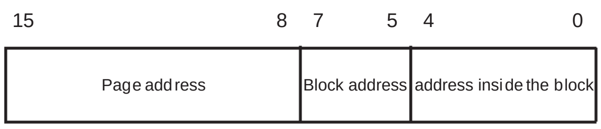


Table 4.1 Sales Type Memory Organization

Device	Flash Memory		RAM Memory		EEPROM Memory	
	Amount	Pages	Amount	Page	Amount	Page(s)
ST52F500x0xx	1024 bytes	0 to 3	128 bytes	32*	-	-
ST52F500x1xx	2048 bytes	0 to 7	128 bytes	32*	-	-
ST52F500x2xx	4096 bytes	0 to 15	256 bytes	32	-	-
ST52F500x3xx	8192 bytes	0 to 31	256 bytes	32	-	-
ST52F503x0xx	896 bytes	0 to 3	128 bytes	32*	128 bytes	3*
ST52F503x1xx	1920 bytes	0 to 7	128 bytes	32*	128 bytes	7*
ST52F503x2xx	3840 bytes	0 to 14	256 bytes	32	256 bytes	15
ST52F503x3xx	7936 bytes	0 to 30	256 bytes	32	256 bytes	31
ST52F504x0xx	4096 bytes	0 to 15	256 bytes	32	512 bytes	16-17
ST52F504x1xx	4096 bytes	0 to 15	256 bytes	32	1024 bytes	16-19
ST52F504x2xx	4096 bytes	0 to 15	256 bytes	32	2048 bytes	16-23
ST52F504x3xx	4096 bytes	0 to 15	256 bytes	32	4096 bytes	16-31

(*) Addresses range from 128 to 255 inside the page

4.2 Memory Programming

The Programming procedure writes the user program and data into the Flash Memory, EEPROM and Option Bytes. The programming procedures are entered by setting the V_{pp} pin equal to V_{dd} and releasing the Reset signal. The following pins are used in Programming mode:

■ V_{PP}	used to switch to programming mode
■ V_{DD}	device supply
■ V_{SS}	device ground
■ RESET	device reset
■ SCL	I^2C serial clock
■ SDA	I^2C serial data

During the device programming, the internal clock is used, so the OSCin and OSCout pins don’t have to be considered.

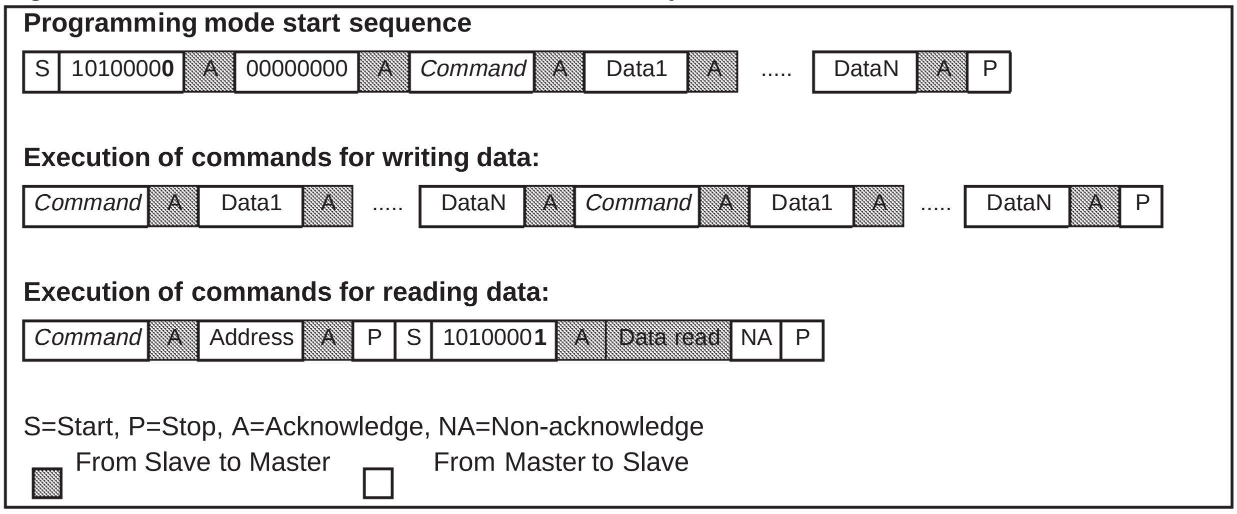
4.2.1 Programming Mode start. The following sequence starts the Programming Mode:

1. V_{PP} is set to V_{DD}
2. The device is Reset (RESET=V_{SS})
3. The Reset is released (RESET=V_{DD})
4. The internal oscillator starts at 10 MHz
5. The memory is turned on
6. The I^2C Interface and Ports are initialized
7. The I^2C Interface is configured to work as Slave, Receiver, 7-bit address and waits for data
8. The Start signal is sent to the chip followed by the Slave Address 1010000 and the direction bit set to 0 (the addressed slave waits for data). The device sends the acknowledge
9. The Programming Mode code 00000000 is sent and acknowledged
10. A command code is sent to the device
11. The procedure related to the command is executed

Table 4.2 Programming Mode Commands

Command	Code	Data in	Data out	Erase	Description
BlockWrite	00000001	32	-	Yes	Write the currently addressed block with the 32 bytes following the command. The Block locations are erased before being written.
ByteWrite	00000010	2	-	Yes	Write the byte addressed by the next data sent in the currently addressed page.
BlockErase	00000011	1		Yes	Erase the block addressed by the next data sent and inside the currently addressed page.
ByteErase	00000100	1		Yes	Erase the byte addressed by the next data sent and inside the currently addressed page.
ByteRead	00000101	1	1	-	Read the byte addressed by the next data sent and inside the current page. The read data is sent by the device after the re-send of the Slave Address with the R/W bit changed.
GlobalErase	00001001	-	-	Yes	All the memory is erased.
FastBlockWrite	00001011	32	-	No	Write the currently addressed block with the 32 bytes following the command. The Block locations aren’t erased.
SetPage	00001100	1	-	-	The currently addressed page is set with the next data sent.
ReadData	00001101	-	1	-	Read the memory location currently addressed. The read data is sent by the device after the command is acknowledged. The current memory absolute address is post-incremented.
IncBlock	00001111	-	-	-	The current block address is incremented modulo 8 (address 0 follows after address 7 and the Page is post-incremented)
ReadStatus	00010011	-	1	-	This command is followed by a status data byte. Mostly used in error condition and to check if the device is locked

Figure 4.1 Commands and Data Communication Sequences


Programming mode start sequence
S | 10100000 | A | 00000000 | A | Command | A | Data1 | A | ..... | DataN | A | P
Execution of commands for writing data:
Command | A | Data1 | A | ..... | DataN | A | Command | A | Data1 | A | ..... | DataN | A | P
Execution of commands for reading data:
Command | A | Address | A | P | S | 10100001 | A | Data read | NA | P
S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge
From Slave to Master
From Master to Slave

The generic procedure of commands execution, with the data communication in both directions is displayed in Figure 4.1.

***Remark:** the Slave Address 1010000 must be sent after a Stop (i.e. each time the data direction changes, to specify the R/W bit). For example: if a command to send data to the device has been executed, a command for receiving data must be followed by the slave address and the R/W bit must be set to 1. The Programming Mode code doesn't need to be specified again .*

***Warning:** After entering the Programming Mode, the currently pointed address is the Page 48, Block 3, byte 0 (Lock Byte).*

The list of the available commands in Programming Mode is showed in Table 4.2

4.2.2 Fast Programming procedure. The fastest way to program the device memory is the use of the *FastBlockWrite* command. The following procedure can be used to write the memory with a new program and new data, starting from the first memory location:

1. The Programming Mode is entered with the sequence described above
2. The memory is erased (all bits are put to 0) with the *GlobalErase* command. The device holds the SCL line low, releasing it after the command is completed (about 2 ms). This command also unlocks the device if locked.
3. The *FastBlockWrite* command is sent and the device acknowledges it
4. The 32 bytes of data to be written in the first memory Block are sent in a sequence. The device acknowledges each of them
5. After the device acknowledges the 32nd byte, it holds the SCL line until the parallel writing of the 32 byte is completed (about 2.5 ms)
6. The Block Pointer is incremented by sending the *IncBlock* command
7. The procedure is repeated from point 3 until there is data to be sent to the memory

***Note:** the Block Pointer assumes values between 0 to 7 (there are 8 blocks in a page). When the Block Pointer is equal to 7, the IncBlock command puts this pointer to 0 and increments the Page Pointer. The Page Pointer, after page writing is completed, does't have to be incremented in the procedure above described.*

4.2.3 Random data writing. A single byte can be written in a specified memory location by using the following procedure:

1. The Programming Mode is entered with the sequence described in Section 4.2.1
2. The *SetPage* command is sent, followed by the page number where the data should be written
3. The *ByteWrite* command is sent followed by two bytes
4. The first bytes that follows the ByteWrite command is the address inside the pointed page where the data must be written.
5. The second byte is the data to be written
6. The device held the SCL line low until the data is not stored in the memory (about 4.5 ms: 2 ms for erasing and 2.5 for writing)

A similar procedure can be used to write a single block:

1. The *SetPage* command is sent, followed by the page number where the data should be written
2. The *IncBlock* command is sent as many times as the block number inside the page (for example: to address the block 3 the *IncBlock* must be sent 3 times)
3. The *WriteBlock* command is sent followed by the 32 data bytes to be written.
4. After the 32th byte is sent, the device holds the SCL line low until all the data are not stored in the memory (about 4.5 ms: 2 ms for erasing and 2.5 for writing: the same time for a single byte)

The procedures described previously can be repeated as many time as needed, without exiting from Programming Mode or re-sending the Slave Address again.

The commands ByteErase and BlockErase, used instead of ByteWrite and BlockWrite, erase (put all bit to 0) the specified memory location or block.

4.2.4 Option Bytes Programming. The Option Byte addresses cannot be accessed with a sequential procedure like the one described in Section 4.2.2. Actually, the pointers are automatically incremented up to the last block or address in page 31. A further increment sets all the pointers to 0.

The Option Byte addresses (located at page 48, block 0, addresses 0-7) must be accessed with a direct addressing procedure as the one described in Section 4.2.3.

If the Fast Programming procedure is used, it must be followed by a Random Block Writing procedure to program the Option Bytes. The other 24 bytes of the block can be written with dummy values.

Figure 4.2 Programming Procedures

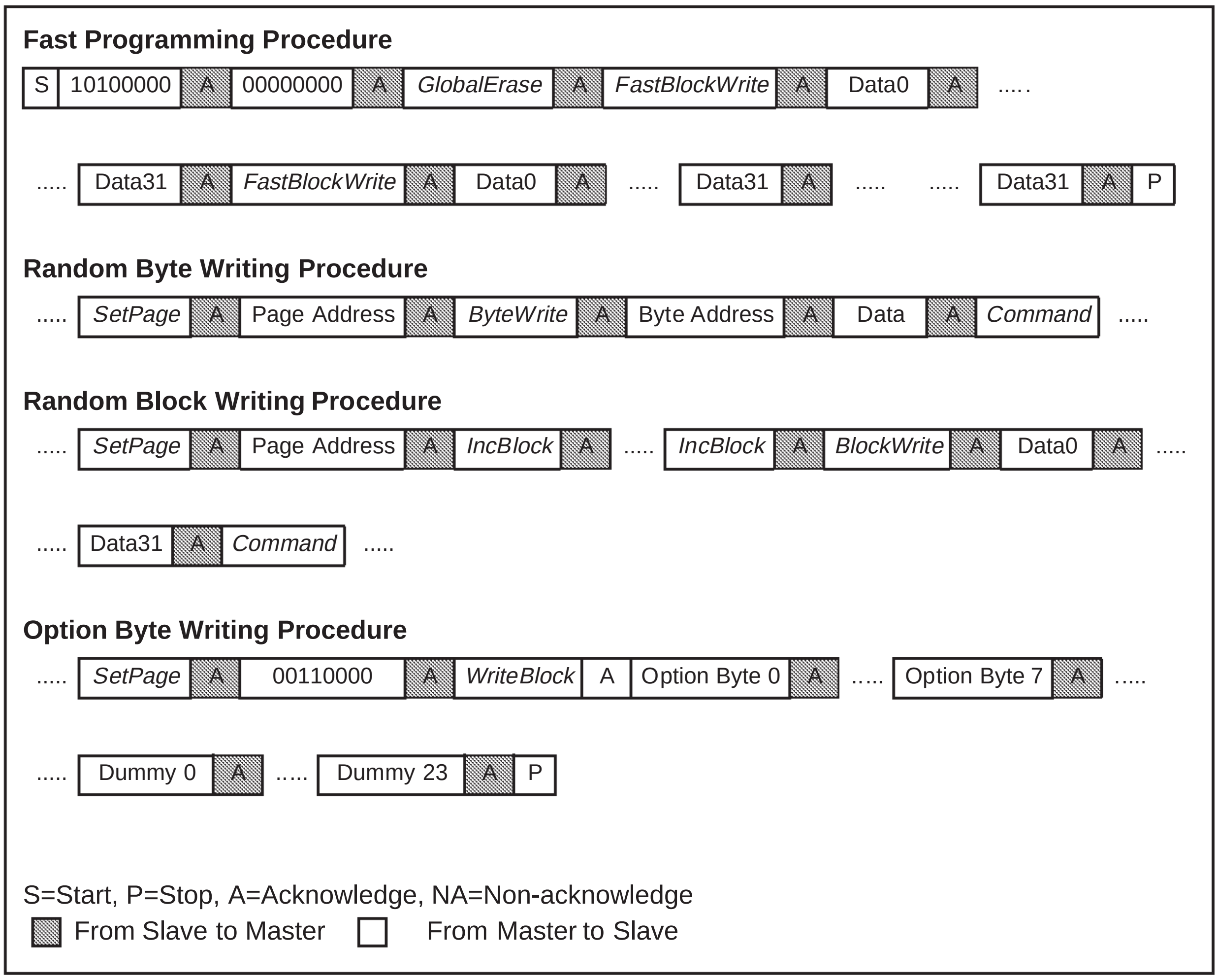
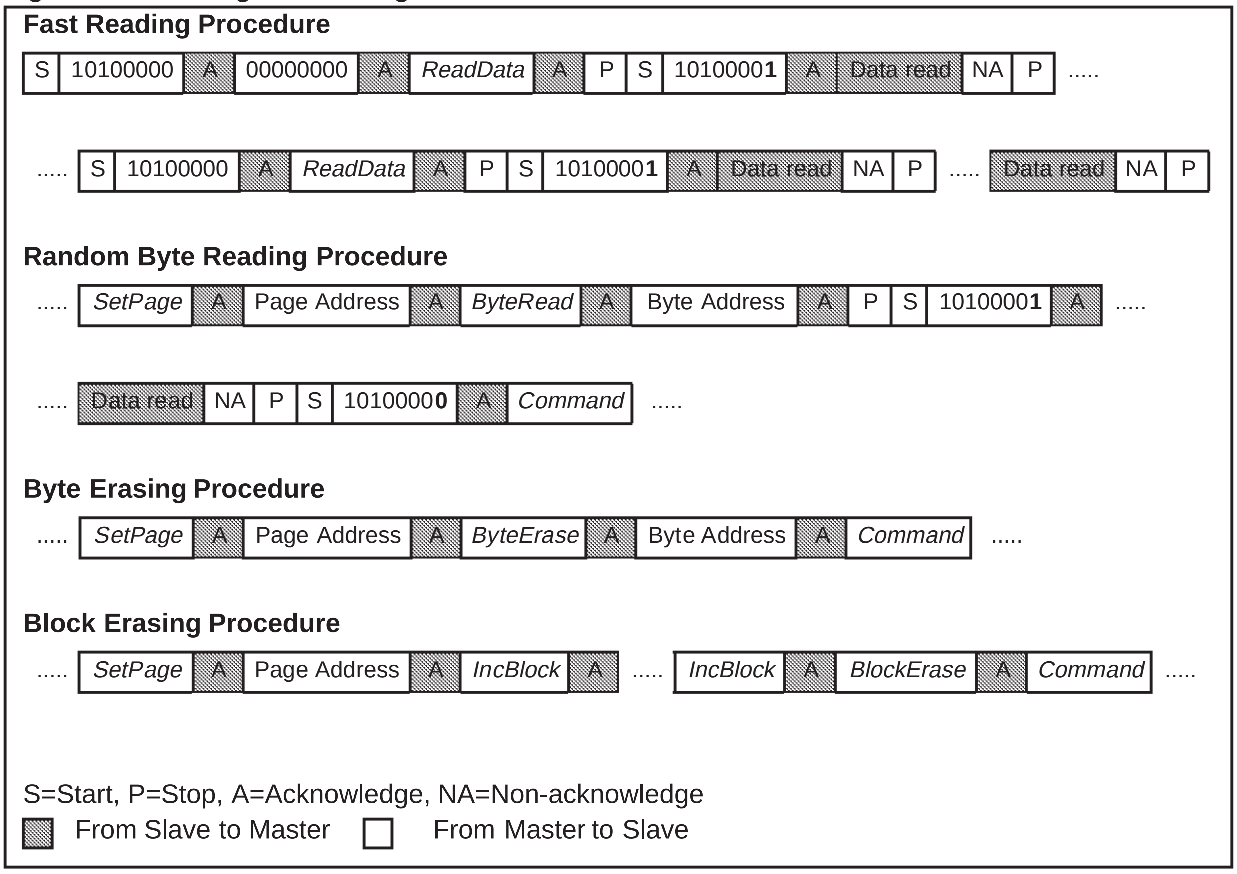

Fast Programming Procedure
S | 10100000 | A | 00000000 | A | GlobalErase | A | FastBlockWrite | A | Data0 | A | .....
..... Data31 | A | FastBlockWrite | A | Data0 | A | ..... Data31 | A | .... Data31 | A | P
Random Byte Writing Procedure
..... SetPage | A | Page Address | A | ByteWrite | A | Byte Address | A | Data | A | Command .....
Random Block Writing Procedure
..... SetPage | A | Page Address | A | IncBlock | A | ..... IncBlock | A | BlockWrite | A | Data0 | A | .....
..... Data31 | A | Command .....
Option Byte Writing Procedure
..... SetPage | A | 00110000 | A | WriteBlock | A | Option Byte 0 | A | ..... Option Byte 7 | A | .....
..... Dummy 0 | A | ..... Dummy 23 | A | P
S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge
From Slave to Master
From Master to Slave

Figure 4.3 Reading and Erasing Procedures


Fast Reading Procedure
S | 10100000 | A | 00000000 | A | ReadData | A | P | S | 10100001 | A | Data read | NA | P | .....
..... | S | 10100000 | A | ReadData | A | P | S | 10100001 | A | Data read | NA | P | ..... | Data read | NA | P
Random Byte Reading Procedure
..... | SetPage | A | Page Address | A | ByteRead | A | Byte Address | A | P | S | 10100001 | A | .....
..... | Data read | NA | P | S | 10100000 | A | Command | .....
Byte Erasing Procedure
..... | SetPage | A | Page Address | A | ByteErase | A | Byte Address | A | Command | .....
Block Erasing Procedure
..... | SetPage | A | Page Address | A | IncBlock | A | ..... | IncBlock | A | BlockErase | A | Command | .....
S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge
From Slave to Master
From Master to Slave

4.3 Memory Verify

To verify the memory contents or just to read part of data stored in memory, the *ByteRead* and the *ReadData* command can be used. The first instruction needs the specification of the address; the second one allows the sequential reading of consecutive memory locations.

Since the device is “Slave” for the I^{2}C protocol, after receiving a command for reading, it must be configured as Slave Transmitter to send the data. In order to do so, the Slave Address (1010000) must be sent again with the R/W byte set to 1, as stated by the communication protocol.

4.3.1 Fast read procedure. The memory can be read sequentially by using the following procedure:

1. The Programming mode is entered with the sequence described in Section 4.2.1
2. The pointers address the memory location 0
3. The *ReadData* command is sent and the device acknowledges it.
4. The Master generates a Stop condition followed by a Start condition
5. The Slave Address with the R/W byte set to 1 (10100001) is sent. The device receives the Slave Address and acknowledges it.
6. The device sends the data to be read in the serial data line SDA. The current absolute address is post-incremented.
7. The Master device doesn’t send the acknowledge and generates a stop condition.
8. To read the next data, the Master generates a Start condition followed by the Slave Address with the R/W byte set to 0 (10100000). The device receives the Slave Address and acknowledges it.
9. The sequence restarts from point 3 until there is data to be read.

Remark: *for the same reasons explained in Section 4.2.4 the Option Bytes cannot be read with this procedure: they can be read with a direct addressing procedure as the one explained in the next section.*

4.3.2 Random data reading. To read a specified memory location, the following procedure should be used:

1. The Programming mode is entered with the sequence described in Section 4.2.1
2. The *SetPage* command is sent, followed to the page number where the data to be read is located
3. The *ByteRead* command is sent, followed by an address inside the page
4. The Master generates a Stop condition followed by a Start condition
5. The Slave Address with the R/W byte set to 1 (10100001) is sent. The device receives the Slave Address and acknowledges it.
6. The device sends the data to be read in the serial data line SDA.
7. The Master device doesn't send the acknowledge and generates a stop condition.
8. To send the next command, the Master should generate a Start condition followed by the Slave Address with the R/W byte set to 0 (10100000).

4.4 Memory Lock

The Program/Data Memory space can be locked to inhibit the reading of contents and protect the intellectual property.

To lock the device, the user must set all the bit of the Lock Byte to '1'. The Lock Byte is located on Page 48 (030h), Block 3, byte 0 inside the block i.e. byte 96 (060h) inside the page.

After writing 255 (0FFh) into the Lock Byte, with the procedure described in the Section 4.2.3, the memory is locked and the only command allowed are the following:

– *GlobalErase*: this command, writing '0' in all the memory, also unlock the device.
– *ReadData*: the only block that can be read is the Block 3 in Page 48 (030h); this allows the reading of the Lock Byte and the ID Code locations (see Section 4.5).
– *ReadStatus*: this command allows the detection of an error condition in Programming mode operation (see Section 4.6). It can also be used to check if the device is locked.

***Remark:** the Lock Byte is checked when entering the Programming Mode. For this reason after writing the Lock Byte, all the commands can be carried out until the Programming mode is exited.*

Figure 4.4 Device Lock Procedure

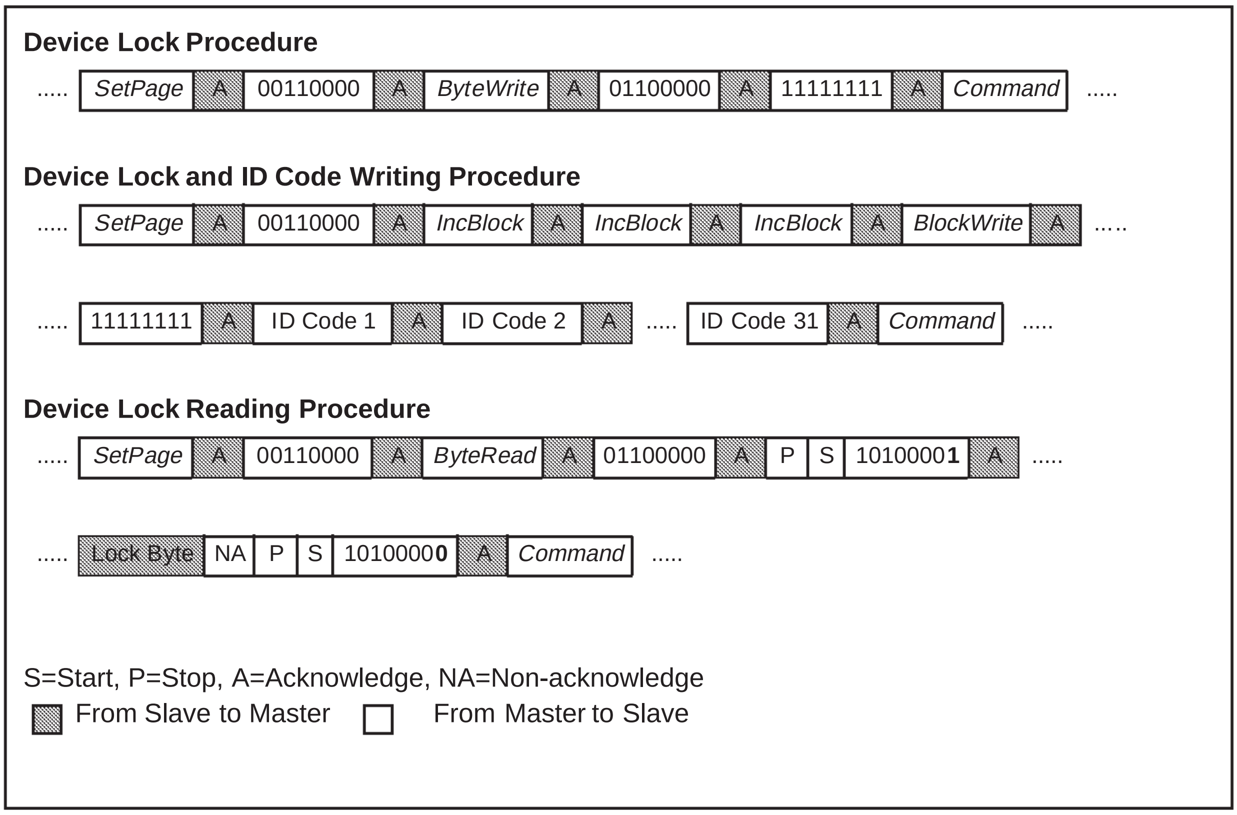

Device Lock Procedure
..... SetPage A 00110000 A ByteWrite A 01100000 A 11111111 A Command .....
Device Lock and ID Code Writing Procedure
..... SetPage A 00110000 A IncBlock A IncBlock A IncBlock A BlockWrite A .....
..... 11111111 A ID Code 1 A ID Code 2 A ..... ID Code 31 A Command .....
Device Lock Reading Procedure
..... SetPage A 00110000 A ByteRead A 01100000 A P S 10100001 A .....
..... Lock Byte NA P S 10100000 A Command .....
S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge
From Slave to Master
From Master to Slave

Figure 4.5 Error Handling Procedure

Wrong command/data case handling:

Wrong Command/Data	▨ A	Command/Data	▨ **NA**	*ReadStatus*	▨ A	P	S	1010000**1**	▨ A	▨ Status Byte	NA	P

S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge

▨ From Slave to Master □ From Master to Slave

When the device is locked, if memory reading is attempted, with the exception of the Lock Byte and ID Code block, the device returns no data and an error sequence.

If memory writing is attempted in any memory location, the device doesn't carry out the command and returns an error sequence.

To unlock the device the *GlobalErase* command must be executed before any writing or reading command.

4.5 ID Code

Block 3 in Page 48 (030h) can also be read if the device is locked. The first byte of the block is the Lock Byte, the other 31 locations are available to the user for writing data, as for example identification codes to distinguish the firmware version loaded in the device.

The ID Code must be written before locking the device: after the device is locked it can only be read. The use of the Block writing procedure is the fastest way: the ID Code is written together the Lock Byte, which is sent first, then the 31 bytes of ID Code follow.

Note: *the ID Code cannot be modified if the device is locked: it can only be read.*

4.6 Error cases

If a wrong command or data is sent to the device, it generates an error condition by not sending the acknowledge after the first successive data or command. Figure 4.5 shows the error sequence.

The error case can be handled by using the *ReadStatus* command. This command can be sent after the error condition is detected; the device returns a Status Byte containing the error code. The *ReadStatus* command sequence is showed in Figure 4.5. The list of the error codes is illustrated in Table 4.3.

Remark: *after the ReadStatus command execution or after any error, the Start Sequence must be carried out before sending a new command.*

The Most Significative Bit of the error codes indicates (when set to '1') that the memory is locked. When a command, that is not allowed when the memory is locked, is sent, the "*Not Allowed"* code is sent. If another code is sent with the MSB to '1' it indicates that the error condition is not caused by the memory lock, but by the event related with the code sent.

Warning: *when the data writing into a non existing location is attempted, no error condition is generated. The user must take care in specifying the correct page address.*

Table 4.3 Error codes

Name	Code	Description
Wrong Direction	x0000001	A transmit direction, not correct in the running sequence, has been set
Stop Missed	x0000010	The Master missed generating a necessary Stop Condition
Data Missing	x0000011	The Master missed to send necessary data to the device
Receive Error	x0000100	The data sent by the Master hasn't been received correctly by the device
Wrong Command	x0000101	The Master sent a wrong command code
Not Allowed	x0000110	A command not allowed when the device is locked has been sent
Wrong Mode	x0010000	A code different form the Programming mode code (00000000) has been sent

4.7 In-Situ Programming (ISP)

The Program/Data Memory can be programmed using the ISP mode. This mode allows the device to be programmed when it is mounted in the user application board.

This feature can be implemented by adding a minimum number of components and board impact.

The programming procedures and pins used are identical to the ones described before for the standard Programming Mode. All the features previously described in this chapter are applicable in ISP mode.

If RESET, SCL and SDA pins are used in the user application board for other purposes, it is recommended to use a serial resistor to avoid a conflict when the other devices force the signal level.

The ISP can be applied by using the standard tools for the device programming. The ST52F500 Starter Kit supplies a cable to perform the ISP. The user application board should supply a suited connector type for the cable (see Starter Kit User Manual).

4.8 In-Application Programming (IAP)

The In Application Programming Mode (IAP) allows the writing of user data in the Flash and EEPROM memories when the user program is running.

There are two ways to write data in IAP mode: single byte write and Block write. Both procedures take about 4.5 ms to complete the writing: the Block write allows the writing of 32 byte in parallel.

Remark: *during data writing, the execution of the user program is stopped until the procedure is completed. Interrupt requests stop the writing operation and the data may be not stored. The bit ABRT in the IAP_SR Input register signals that the data writing hasn't been completed. To assure writing completion, the user should globally disable the interrupts (UDGI instruction) before starting IAP data writing.*

4.8.1 Single byte write. Writing of a single byte in the Non-Volatile Program/Data memory is performed by using the LDER instruction (both direct and indirect addressing). The memory page should be indicated before the LDER instruction with the PGSET or PGSETR instruction. The byte address inside the page is specified by the LDER instruction itself.

As soon as the instruction is executed, the data writing starts and is performed in about 4.5 ms.

4.8.2 Block write. This procedure allows the writing of 32 bytes in parallel. These bytes should belong to the same block.

Before the writing in the Program/Data memory, data must be buffered in the Register File in the first 32 locations (0-31, 00h-020h) by using the normal instructions to load the Register File locations.

Then the data writing starts by using the BLKSET instruction. The destination block is addressed by specifying the memory page with the PGSET or PGSETR instruction before to start the writing; the block inside the page is addressed with the argument of the BLKSET instruction.

Example:

```
PGSET 5
BLKSET 4
```

This instruction sequence writes the contents of the first 32 bytes of the Register File in the locations 1152-1183 (0480h-049Fh).

Warning: *the user should be careful in specifying the correct page and block: the addressing of an not existing block can cause the unwanted writing of a different block.*

As soon as the BLKSET instruction is executed, the data writing starts and is performed in about 4.5 ms.

This procedure may also be used to write few data, taking in account that all the 32 byte are written in the block anyway.

4.8.3 Memory Corruption Prevention.

The user can protect some pages (or all the memory) from unintentional writings. The only constraint is that the protected pages must be consecutive.

Two Option Bytes allow the specification of the page to be protected: PG_LOCK (Option Byte 5) and PG_UNLOCK (Option Byte 6). PG_LOCK is used to specify the first protected page; PG_UNLOCK is used to specify the first page not protected after the protected ones. The pages between the two addresses are protected.

When writing in a protected page is attempted, the procedure is aborted and the bit PRTCD of IAP_SR Input register is set.

If the PG_LOCK and PG_UNLOCK have the same value, no page is protected. By default, the two Option Bytes are programmed with the value 0, so the memory is not write protected by default.

In Programming Mode the protection is not considered and the pages can be written unless the device is locked.

4.8.4 Option Bytes.

First Protected Page (PG_LOCK)
Option Byte 5 (05h)
Reset Value: 0000 0000 (00h)

7							0
LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0

Bit 7-0: **LCK7-0** First Page write protected

In this register the address of first page to be protected in writing is specified. The pages following this one are protected up to the page specified by the PG_UNLOCK Option Byte (not included among the protected ones).

First Page not Protected (PG_UNLOCK)
Option Byte 6 (06h)
Reset Value: 0000 0000 (00h)

7							0
UNLCK7	UNLCK6	UNLCK5	UNLCK4	UNLCK3	UNLCK2	UNLCK1	UNLCK0

Bit 7-0: **UNLCK7-0** First Page not write protected

In this register the address of first page not write protected after the protected ones is specified. The pages following this one aren't protected.

4.8.5 Input Register.

IAP Status Register (IAP_SR)
Input Register 40 (028h) Read only
Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	-	-	PRTCD	ABRT

Bit 7-2: Not Used

Bit 1: **PRTCD** Page Protected
0: The writing has been completed
1: The writing has been aborted because the page is protected.

Bit 0: **ABRT** Writing operation aborted
0: The writing has been completed
1: The writing has been aborted because an interrupt or another unspecified cause occurred.

The ABRT and PRTCD bits are reset after the next successful data writing in the Flash of EEPROM memory.

5 INTERRUPTS

The Control Unit (CU) responds to peripheral events and external events through its interrupt channels.

When such events occur, if the related interrupt is not masked and doesn't have a priority order, the current program execution can be suspended to allow the CU to execute a specific response routine.

Each interrupt is associated with an interrupt vector that contains the memory address of the related interrupt service routine. Each vector is located in the Program/Data Memory space at a fixed address (see Figure 3.2 Program/Data Memory Organization).

5.1 Interrupt Processing

If interrupts are pending at the end of an arithmetic or logic instruction, the interrupt with the highest priority is acknowledged. When the interrupt is acknowledged the flags and the current PC are saved in the stacks and the associated Interrupt routine is executed. The start address of this routine (Interrupt Vector) is located in three bytes of the Program/Data Memory between address 3 and 32 (03h-020h). See Table 5.1 for the list of the Interrupt Vector addresses.

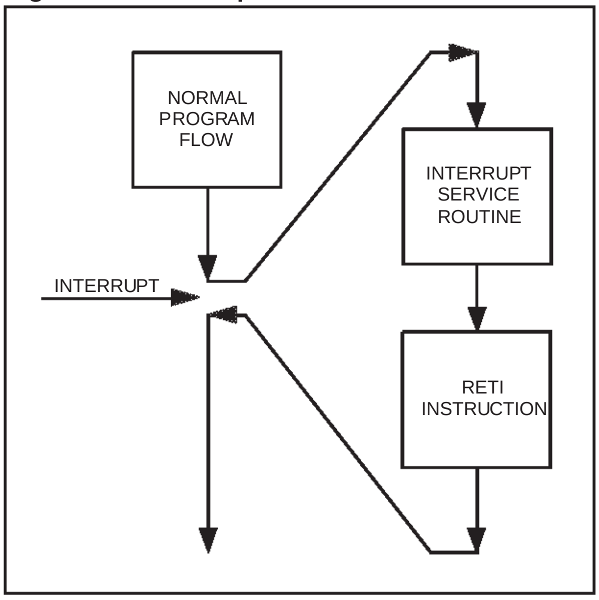
The Interrupt routine is performed as a normal code. At the end of each instruction, the CU checks if a higher priority interrupt has sent an interrupt request. An Interrupt request with a higher priority stops lower priority Interrupts. The Program Counter and the flags are stored in their own stacks.

With the instruction RETI (Return from Interrupt) the flags and the Program Counter (PC) are restored from the top of the stacks. These stacks have already been described in Paragraph 3.4.

An Interrupt request cannot stop fuzzy rule processing, but only after the end of a fuzzy rule or at the end of a logic or arithmetic instruction, unless a Global Interrupt Disable instruction has been executed before (see below).

Remark: *A fuzzy routine can be interrupted only in the Main program. When a Fuzzy function is running inside another interrupt routine an interrupt request can cause side effects in the Control Unit. For this reason, in order to use a Fuzzy function inside an interrupt routine, the user MUST include the Fuzzy function between an UDGI (MDGI) instruction and an UEGI (MEGI) instruction (see the following paragraphs), in order to disable the interrupt request during the execution of the fuzzy function.*

Figure 5.1 Interrupt Flow



5.2 Global Interrupt Request Enabling

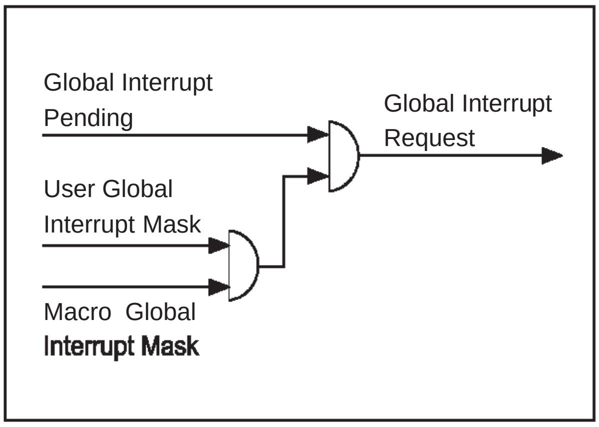
When an Interrupt occurs, it generates a Global Interrupt Pending (GIP). After a GIP a Global Interrupt Request (GIR) will be generated and Interrupt Service Routine associated with the interrupt with higher priority will start.

In order to avoid possible conflicts between the interrupt masking set in the main program, or inside high level language compiler macros, the GIP is put in AND through the User Global Interrupt Mask or the Macro Global Interrupt Mask (see Figure 5.2).

The UEGI/UDGI instruction switches the User Global Interrupt Mask enabling/disabling the GIR for the main program.

MEGI/MDGI instructions switch the Macro Global Interrupt Mask on/off in order to ensure that the macro will not be interrupted.

Figure 5.2 Global Interrupt Request



5.3 Interrupt Sources

ST FIVE manages interrupt signals generated by the internal peripherals or generated by software by the TRAP instruction or coming from the Port pins. There are two kinds of interrupts coming from the Port pins: the NMI and the Ports Interrupts.

NMI (Not Maskable Interrupt) is associated with pin PA7 when it is configured as Alternate Function. This interrupt source doesn't have a configurable level priority and cannot be masked. The fixed priority level is lower than the software TRAP and higher than all the other interrupts. The NMI can be configured to be active on the rising or the falling edge.

The Port Interrupts sources are connected with Port A and Port B pins. The pins belonging to the same Port are associated with the same interrupt vector: there is one vector for Port A and one for Port B. In order to use one port pin as interrupt, it must be configured as an interrupt source (see I/O Ports chapter). In this manner, up to 16 Port Interrupt sources are available. By reading the Port the sources that belong to the same Port can be discriminated. The Port Interrupts can be configured to be active on the rising or the falling edge.

All the interrupt sources are filtered, in order to avoid false interrupt requests caused by glitches.

The Trap instruction is something between a interrupt and a call: it generated an interrupt request at top priority level and the control is passed to the associated interrupt routine which vector is located in the fixed addresses 30-32. This routine cannot be interrupted and it is serviced even if the interrupts are globally disabled.

Note: *Similarly to the CALL instruction, after a TRAP the flags are not stacked.*

5.4 Interrupt Maskability and Priority Levels

Interrupts can be masked by the corresponding INT_MASK Configuration Register 0 (00h). An interrupt is enabled when the mask bit is "1". Vice versa, when the bit is "0", the interrupt is masked and the eventual requests are kept pending.

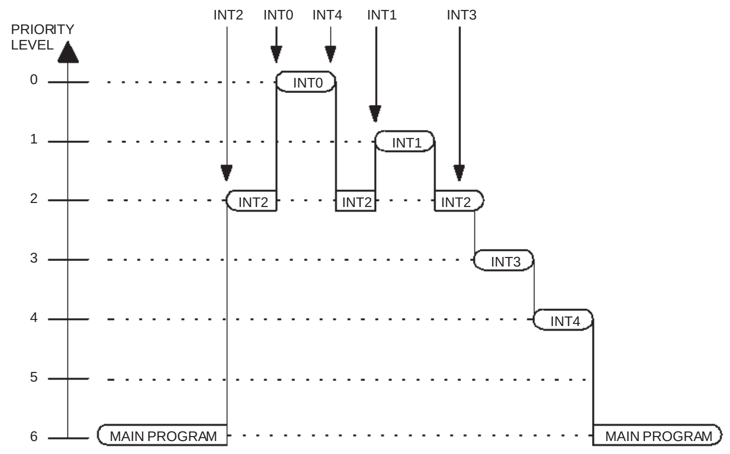
All the interrupts, with the exception of the NMI and TRAP that have fixed level priority, have a configurable priority level. The configuration of the priority levels is completed by writing three consecutive Configuration Registers: INT_PRIORITY_H, INT_PRIORITY_M, INT_PRIORITY_L, addresses from 2 to 4 (02h-04h). The 24 bits of these registers are divided into 8 groups of three bits: each group is associated with a priority level. The three bits of each group are written with the code number associated with the interrupt source. See Table 5.1 to know the codes.

Remark: *The priority levels Configuration Registers must be programmed with different values for each 3-bit groups to avoid erroneous operation. For this reason the Interrupt priority must be fixed at the beginning of the main program, because the reset values of the Configuration Registers correspond to an undefined configuration (all zeros). During program execution the interrupt priority can only be modified within the Main Program: it cannot be changed within an interrupt service routine.*

5.5 Interrupt RESET

When an interrupt is masked, all requests are not acknowledged and remain pending. When the pending interrupt is enabled it is immediately serviced. This event may be undesired; in order to avoid this a RINT instruction may be inserted followed by the code number that identifies the interrupt to reset the pending request. See Table 5.1 to know the codes.

Figure 5.3 Example of Interrupt Requests



5.6 Register Description

Interrupt Mask Register (INT_MASK)
Configuration Register 0 (00h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
MSKPB	MSKPA	MSKI2C	MSKSPI	-	MSKT1	MSKT0	-

Bit 7: **MSKPB** Interrupt Mask Port B
0: Port B interrupt masked
1: Port B interrupt enabled

Bit 6: **MSKPA** Interrupt Mask Port A
0: Port A interrupt masked
1: Port A interrupt enabled

Bit 5: **MSKI2C** Interrupt Mask I^2C Interface
0: I^2C Interface interrupt masked
1: I^2C Interface interrupt enabled

Bit 4: **MSKSPI** Interrupt Mask SPI
0: SPI interrupt masked
1: SPI interrupt enabled

Bit 3: Not used

Bit 2: **MSKT1** Interrupt Mask PWM/Timer 1
0: Pwm/Timer 1 interrupt masked
1: Pwm/Timer 1 interrupt enabled

Bit 1: **MSKT0** Interrupt Mask Pwm/Timer 0
0: Pwm/Timer 0 interrupt masked
1: Pwm/Timer 0 interrupt enabled

Bit 0: Not used

Interrupt Polarity Register (INT_POL)
Configuration Register 1 (01h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	LVD_EN	T0RPOL	RESPOL	STRPOL	POLPA	POLNMI

Bit 7-6: Not Used

Bit 4-3: See Timer 0 Registers Description

Bit 2: **POLPB** Port B Interrupt Polarity
0: The Port B interrupt is triggered on the rising edge of the applied external signal.
1: The Port B interrupt is triggered on the falling edge of the applied external signal.

Bit 1: **POLPA** Port A Interrupt Polarity
0: The Port A interrupt is triggered on the rising edge of the applied external signal.
1: The Port A interrupt is triggered on the falling edge of the applied external signal.

Bit 0: **POLNMI** Non Maskable Interrupt Polarity
0: The NMI is triggered on the rising edge of the applied external signal.
1: The NMI is triggered on the falling edge of the applied external signal.

High Priority Register (INT_PRL_H)
Configuration Register 2 (02h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
PRL23	PRL22	PRL21	PRL20	PRL19	PRL18	PRL17	PRL16

Medium Priority Register (INT_PRL_M)
Configuration Register 3 (03h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
PRL15	PRL14	PRL13	PRL12	PRL11	PRL10	PRL9	PRL8

Low Priority Register (INT_PRL_L)
Configuration Register 4 (04h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
PRL7	PRL6	PRL5	PRL4	PRL3	PRL2	PRL1	PRL0

These three register are used to configure the priority level of each interrupt source. The 24 bits of these registers (PRL24-PRL0) are divided into 8 groups of three bits: each group is associated with a priority level (from level 1, the highest, to level 8, the lowest: level 0 is fixed for the NMI that can be interrupted only by the TRAP) . The three bits of each group are written with the code number associated with the interrupt source (see Table 5.1).

PRL2-PRL1: Interrupt priority level 1 (highest)

PRL5-PRL3: Interrupt priority level 2

PRL8-PRL6: Interrupt priority level 3

PRL11-PRL9:Interrupt priority level 4

PRL14-PRL12: Interrupt priority level 5

PRL17-PRL15: Interrupt priority level 6

PRL20-PRL18: Interrupt prioritylevel 7

PRL23-PRL21: Interrupt priority level 8 (lowest)

Example: writing the code 110 into PRL8-PRL6 bits the priority level 3 is assigned to the Port A Interrupt.

Warning: *the Priority Level configuration registers must be always configured.*

Table 5.1 Interrupt sources paramethers

Interrupt Source	Priority type	PRL code	RINT code	Maskable	Vector Addresses
PWM/Timer 0	Programmable	001	1	Yes	6-8 (06h-08h)
PWM/Timer 1	Programmable	010	2	Yes	9-11 (09h-0Bh)
SPI	Programmable	100	4	Yes	15-17 (0Fh-011h)
I^{2}C Interface	Programmable	101	5	Yes	18-20 (012h-014h)
Port A	Programmable	110	6	Yes	21-23 (015h-017h)
Port B	Programmable	111	7	Yes	24-26 (018h-01Ah)
NMI	Fixed	-	8	No	27-29 (01Bh-01Dh)
TRAP	Fixed to highest	-	-	No	30-32 (01Eh-020h)

6 CLOCK, RESET & POWER SAVING MODES

6.1 Clock

The ST52F500/F503/F504 Clock Generator module generates the internal clock for the internal Control Unit, ALU and on-chip peripherals. The Clock is designed to require a minimum of external components.

ST52F500/F503/F504 devices supply the internal oscillator in four clock modes:

- External oscillator
- External clock
- External RC oscillator
- Internal clock

The device always starts in internal clock mode, excluding any external clock source. After the start-up phase the clock is configured according to the user definition programmed in the Option Bytes 0 (OSC_CR). The internal clock generator can supply an internal clock signal with a fixed frequency of 10 MHz ± 1%, without the need for external components. In order to obtain the maximum accuracy, the frequency can be calibrated by configuring the related Option byte 2 (OSC_SET).

The external oscillator mode uses a quartz crystal or a ceramic resonator connected to OSCin and OSCout as illustrated in Figure 6.1. This figure also illustrates the connection of an external clock.

The ST52F500/F503/F504 oscillator circuit generates an internal clock signal with the same period and phase as the OSCIN input pin. The maximum frequency allowed is 24 MHz.

When the external oscillator is used, the loop gain can be adapted to the various frequencies values by configuring the three bits of the Option Byte 1 CLK_SET (see Register Decription, Table 6.2).

When an external clock is used, it must be connected to the pin OSCIN while OSCOUT can be floating. In this case, Option Byte 1 bits must be written with 0 (000).

The crystal oscillator start-up time is a function of many variables: crystal parameters (especially R_S), oscillator load capacitance (CL), IC parameters, environment temperature and supply voltage.

The crystal or ceramic leads and circuit connections must be as short as possible. Typical values for CL1, CL2 are 10pF for a 20 MHz crystal.

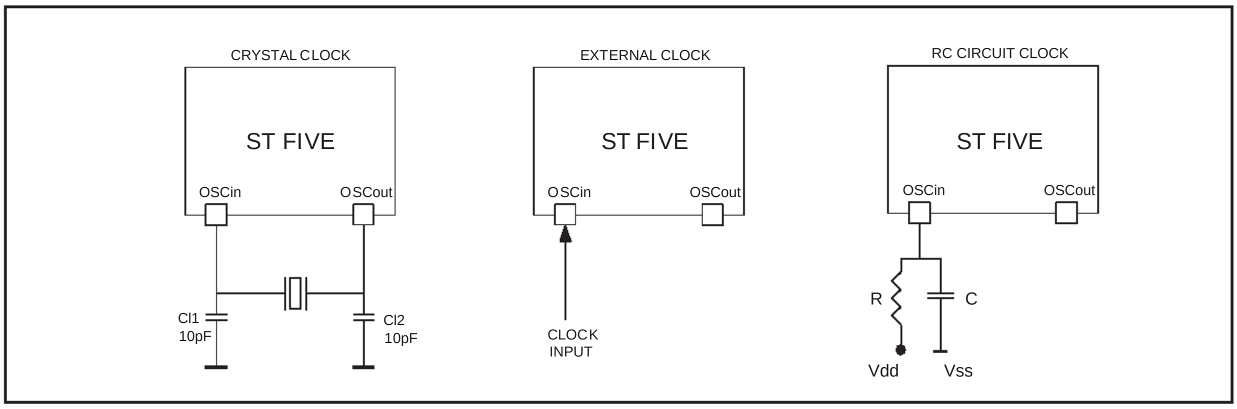
The clock signal can also be generated by an external RC circuit offering additional cost savings. Figure 6.1 illustrates the possible connections. Frequency is a function of resistor, capacitance, supply voltage and operating temperature; some indicative values when Vdd=5V and T=25°, are shown in Table 6.1.

The clock signal generates two internal clock signals: one for the CPU and one for the peripherals. The CPU clock frequency can be reduced, in order to decrease current consuption, by setting the CPU_CLK Configuration Register 46 (02Eh). The CPU clock can be reduced up to 64 times (see Register Description).

Table 6.1 RC Oscillator indicative frequencies

C (pF)	R(Ω)	f_{osc} (KHz)	Variation
20 pF	9.5K	5000	6.6%
	10K	4870	7.1%
	20K	3000	5.3%
	50K	1360	3.3%
	100K	724	2.8%
100 pF	10K	1720	7.5%
	20K	926	8%
	50K	424	11.2%
	100K	248	15%

Figure 6.1 Oscillator Connections



6.2 Reset

Four Reset sources are available:

- RESET pin (external source)
- WATCHDOG (internal source)
- POWER ON Reset (Internal source)
- PLVD Reset (Internal source)

When a Reset event occurs, the user program restarts from the beginning.

6.2.1 External Reset. Reset is an input pin. An internal reset does not affect this pin. A Reset signal originated by external sources is recognized immediately. The RESET pin may be used to ensure Vdd has risen to a point where the ICU can operate correctly before the user program is run. Reset must be set to Vdd in working mode.

A Pull up resistor of 100 KΩ guarantees that the RESET pin is at level “1” when no HALT or Power-On events occur. If an external resistor is connected to the RESET pin a minimum value of 10KΩ must be used.

6.2.2 Reset Procedures. After the Reset pin is set to Vdd or following a Power-On Reset event, the device is not started until the internal supply voltage has reached the nominal level of 2.5 V (corresponding roughly to Vdd=2.8 V).

After this level has been reached, the internal oscillator (10 MHZ) is started and a delay period of 4.096 clock cycles is initiated, in order to allow the oscillator to stabilize and to ensure that recovery has taken place from the Reset state.

If the device has been configured to work with the internal clock, the user program is started, otherwise the Option Byte 7 (WAKEUP) is read and another count is started before running the user program. The count duration depends on the contents of the Option Byte 7 (WAKEUP), that works as a prescaler, according to the follwing formula:

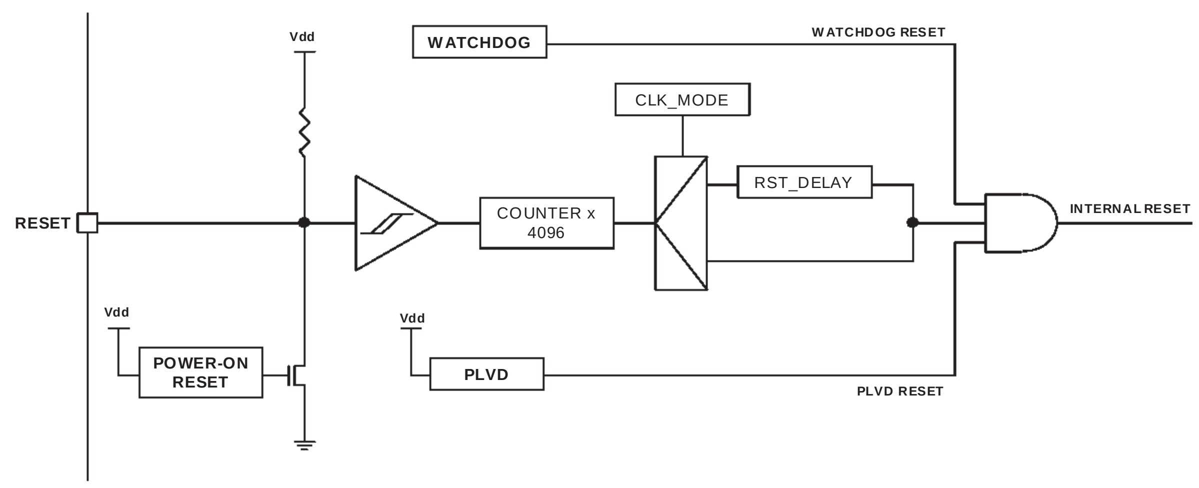
$$Delay = 4096 \times (WAKEUP + 1) \times Tclk$$

This delay has been introduced in order to ensure that the oscillator has become stable after its restart.

If the Reset is generated by the PLVD or the Watchdog, the oscillator is not turned off; for this reason the CPU is then restarted immediately, without the delay.

After a RESET procedure is completed, the core reads the instruction stored in the first 3 bytes of the Program/Data Memory, which contains a JUMP instruction to the first instruction of the user program. The Assembler tool automatically generates this Jump instruction with the first instruction address.

Figure 6.2 Reset Block Diagram



6.3 Programmable Low Voltage Detector

The on-chip Programmable Low Voltage Detector (PLVD) circuit prevents the processor from falling into an unpredictable status if the power supply drops below a certain level.

When Vdd drops below the detection level, the PLVD causes an internal processor Reset that remains active as long as Vdd remains below the trigger level.

The PLVD resets the entire device except the Power-on Detector and the PLVD itself.

The PLVD can be enabled/disabled at reset by setting the Option Byte 2 (PLVD_CR) bits.

When Vdd increases above the Trigger Level, the PLVD reset is deactivated and the user program is started from the beginning.

The detection levels are programmable by means of the Option Byte 2 (PLVD_CR). There are three levels for the PLVD falling voltages (2.9V, 3.4V, 3.9V) and for rising voltages (3.1V, 3.65V, 4.2V). The hysteresis for each level are respectively 200 mV, 250 mV and 300 mV.

The PLVD circuit will only detect a drop if Vdd voltage stays below the safe threshold for at least 5µs before activation/deactivation of the LVD in order to filter voltage spikes.

The PLVD function isn't active when it is in HALT mode.

6.4 Power Saving modes

There are two types of Power Saving modes: WAIT and HALT mode. These conditions may be entered by using the WAIT or HALT instructions.

6.4.1 Wait Mode. Wait mode places the ICU in a low power consumption status by stopping the CPU. All peripherals and the watchdog remain active. During WAIT mode the Interrupts are enabled. The ICU remains in Wait mode until an Interrupt or a RESET occurs, whereupon the Program Counter jumps to the interrupt service routine or, if a Reset occurs, to the beginning of the user program.

6.4.2 Halt Mode. Halt mode is the lowest ICU power consumption mode, which is entered by executing the HALT instruction. The internal oscillator is turned off, causing all internal processing to be terminated, including the operations of the on-chip peripherals. Halt mode cannot be used when the watchdog is enabled. If the HALT instruction is executed while the watchdog system is enabled, it will be skipped without modifying the normal CPU operations.

The ICU can exit Halt mode upon reception of an NMI, a Port Interrupt or a Reset. The internal oscillator (10 MHZ) is started and a delay period of 4.096 clock cycles is initiated, in order to allow the oscillator to stabilize and to ensure that recovery has taken place from the Reset state.

If the device has been configured to work with the internal clock, the user program is started, otherwise the Option Byte 7 (WAKEUP) is read and another count is started before running the user program. The count duration depends on the contents of the Option Byte 7 (WAKEUP), that works as prescaler, according to the follwing formula:

$$Delay = 4096 \times (WAKEUP + 1) \times Tclk$$

This delay has been introduced in ordet to ensure that the oscillator has become stable after it is restarted.

After the start up delay, by exiting with the NMI or a Port interrupt, the CPU restarts operations by serving the associated interrupt routine.

Figure 6.3 WAIT Flow Chart

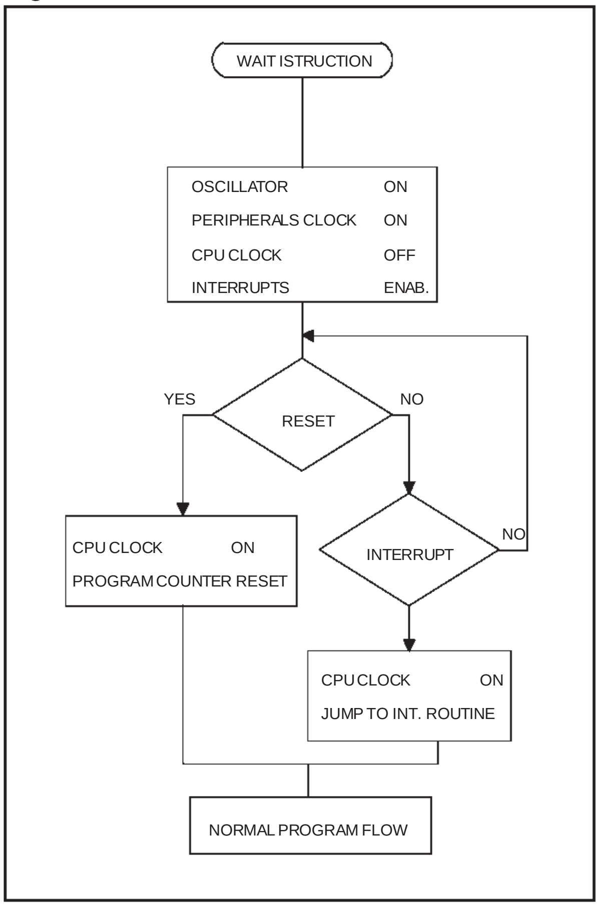
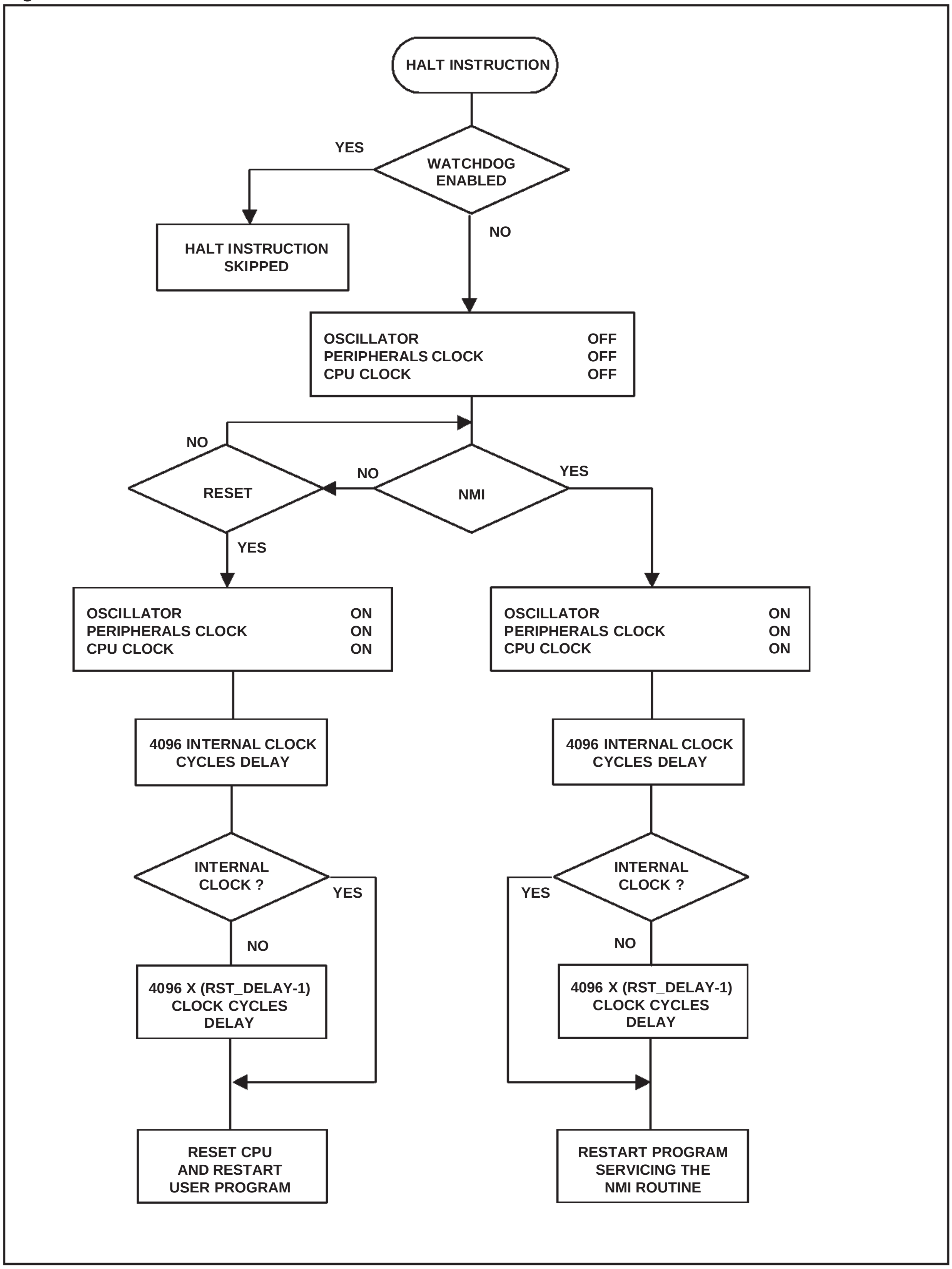


Figure 6.4 HALT Flow Chart



6.5 Register Description

The following section describes the Register which are used to configure the Clock, Reset and PLVD.

6.5.1 Configuration Register.

CPU Clock Prescaler (CPU_CLK)

Configuration Register 46 (02Eh) Read/Write

Reset Value: 0000 0000 (00h)

7							0
-	-	CPUCK5	CPUCK4	CPUCK3	CPUCK2	CPUCK1	CPUCK0

Bit 7-6: Not Used

Bit 5-0: **CPUCK5-0** CPU Clock Prescaler bits

The CPU Clock frequency is divided by a factor described in the following table

CPUCK5-0	CPU Clock
000000	$f_{CPU}=f_{OSC}$
000001	$f_{CPU}=f_{OSC}/2$
000010	$f_{CPU}=f_{OSC}/4$
000100	$f_{CPU}=f_{OSC}/8$
001000	$f_{CPU}=f_{OSC}/16$
010000	$f_{CPU}=f_{OSC}/32$
100000	$f_{CPU}=f_{OSC}/64$
others	$f_{CPU}=f_{OSC}/64$

6.5.2 Option Bytes.

Clock Mode (OSC_CR)

Option Byte 0 (00h)

Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	-	-	CKMOD1	CKMOD0

Bit 7-2: Not Used

Bit 1-0: **CKMOD1-0** Clock Mode

00: Internal Oscillator

01: External Clock or quartz

1x: External RC oscillator

External Clock Parameters (CLK_SET)

Option Byte 1 (01h)

Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	-	CKPAR2	CKPAR1	CKPAR0

Bit 7-3: Not Used

Bit 2-0: **CKPAR2-0** Oscillator Gains

These three bits enable/disable the loop gains when a external clock or quartz are used for generating the clock. The following table decribes the possible configuration options. Table 6.2 illustrates the reccomended values for the most common frequencies used, time to start the oscillations and the settling time to have a duty cycle of 40%-60% (at steady state it is 50%).

CKPAR2-0	Enabled Gain Stages
000	No Gains (External Clock Mode)
001	1 gain stage enabled
010	not allowed
011	3 gain stage enabled
100	not allowed
101	4 gain stage enabled
110	not allowed
111	6 gain stage enabled

Warning: *If an External Clock is used instead of a quartz or ceramic resonator, it is reccomended that no gain be enabled (CKPAR2-0=000) in order lo lower the current consuption.*

Table 6.2 Recomended Gains for the most common frequencies

Frequency	Recommend Gain Stages	CKPAR2-0	Oscillation Start Times*	Settling Times for 40%-60% duty-cycle*
External Clock	0	000	-	-
5 MHz	1	001	100 µs	110 µs
10 MHz	3	011	80 µs	85 µs
20 MHz	6	111	133 µs	143 µs

(*) Values by design (not characterized)

Internal Oscillator Calibration (OSC_SET)

Option Byte 2 (02h)

Reset Value: 0000 0000 (00h)

7							0
-	-	OSPAR5	OSPAR4	OSPAR3	OSPAR2	OSPAR1	OSPAR0

Bit 7-6: Not Used

Bit 5-0: **OSPAR5-0** Internal Oscillator Parameters
These bits are used in order to calibrate the precision of the internal oscillator working at 10 MHz. The six bits enable some current generators with steps of 5 µA corresponding to interval of frequency of 100KHz.

Warning: *the maximum configuration value allowed is 101000 (40). The value coresponding to the 10 MHz by design is 010100 (20).*

PLVD Control Register (PLVD_CR)

Option Byte 3 (03h)

Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	-	-	PLVD1	PLVD0

Bit 7-2: Not Used

Bit 1-0: **PLVD1-0** PLVD detection levels
00: PLVD disabled
01: Lowest detection level
10: Medium detection level
11: Highest detection level

Wake-Up Time Prescaler (WAKEUP)

Option Byte 7 (07h)

Reset Value: 0000 0000 (00h)

7							0
WK7	WK6	WK5	WK4	WK3	WK2	WK1	WK0

Bit 7-0: **WK7-0** Wake-up prescaler
This byte determinates the time delay for the stabilization of the oscillator after an External Reset or a POR and after the wake-up from Halt. The time delay is computed according to the following formula:

$$Delay = 4096 \times (WAKEUP + 1) \times Tclk$$

Warning: *the value 255 for WAKEUP is not allowed. If the internal clock is used as clock source the prescaler is not used.*

7 I/O PORTS

7.1 Introduction

ST52F500/F503/F504 are characterized by flexible individually programmable multi-functional I/O lines. The ST52F500/F503/F504 supplies devices with up to 3 Ports (named from A to C) with up to 22 I/O lines.

Each pin can be used as a digital I/O or can be connected with a peripheral (Alternate Function). The I/O lines belonging to Port A and Port B can also be used to generate Port Interrupts.

The I/O Port pins can be configured in the following modes:

- Input high impedance (reset state)
- Input with pull-up
- Output with pull-up
- Output push-pull
- Output with weak pull-up
- Output open drain
- Interrupt with pull-up
- Interrupt without pull-up

These eight modes can be selected by programming three Configuration Registers for each Port. All the pins that belong to the same Port can be configured separately by setting the corresponding bits in the three registers (see Register Description).

To avoid side effects, the Configuration Registers register are latched only when the Direction Register (PORT_x_DDR) is written. For this reason this register must be always written when modifying the pin configuration.

The I/O digital pins (Port A, C) are TTL compatible and have a Schmitt Trigger. The output buffer can supply high current sink (up to 8mA).

7.2 Input Mode

The pins configured as input can be read by accessing the corresponding Port Input Register by means of the LDRI instruction. The addresses for Port A , B and C are respectively 0 (00h), 1 (01h), and 2 (02h).

When executing the LDRI instruction all the signals connected to the input pins of the Port are read and the logical value is copied in the specified Register File location. If some pins are configured in output, the port buffer contents, which are the last written logical values in the output pins, are read.

7.3 Output Mode

The pins configured as output can be written by accessing the corresponding Port Output Register by means of the LDPR, LDPI and LDPE instructions. The addresses for Port A , B and C are respectively, 0 (00h), 1 (01h), and 2 (02h).

When executing the above mentioned instructions, the Port buffer is written and the Port pin signals are modified. If some pins are configured as input or as interrupt, the values are ignored.

7.4 Interrupt Mode

The pins configured as Interrupt Mode can generate a Port Interrupt request. Only Port A and Port B pins can be configured in this mode.

An Interrupt vector is associated to each Port: there are two Port Interrupts available but more pins of the ports can act as source at the same time.

The Configuration Registers switch the signals deriving from interrupt pins to an OR gate that generates the interrupt request signal. The signal deriving from the pins can be read, allowing the discrimination of the interrupt sources when more than one pin can generate the interrupt signal.

The interrupt trigger can be configured either in the rising or falling edge of the external signal.

Figure 7.1 Digital Pin

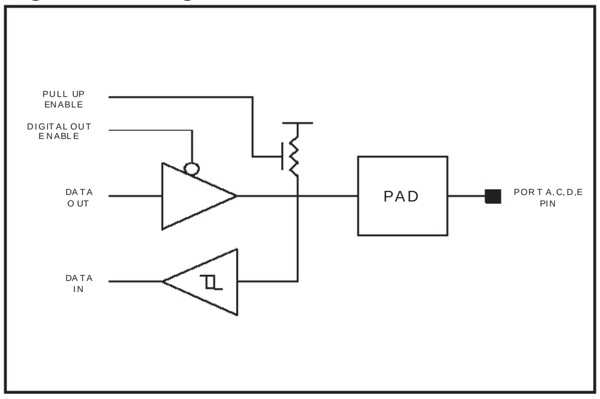
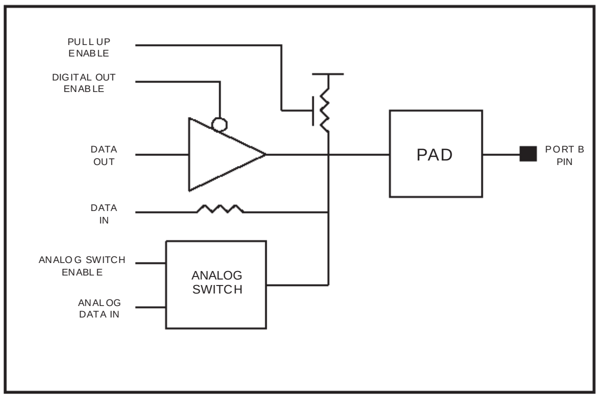


Figure 7.2 Analog Pin



7.5 Alternate Functions

The Alternate Function allows the pins to be connected with the peripheral signals or NMI. Not all Port pins have an Alternate Function associated.

A Configuration Register (PORT_x_AF) for each Port is used to switch from the Digital I/O function or the Alternate Function.

Some pins can have two Alternate Functions: one input function and one output function. To switch between the two functions, the PORT_x_AF must be configured in Alternate Function mode and the PORT_x_DDR Configuration Register must be switched in Input mode or in Output mode.

NMI is considered an Alternate Function. For this reason an NMI interrupt request can't be generated unless the PA7 pin is configured in Alternate Function and in one of the Input modes.

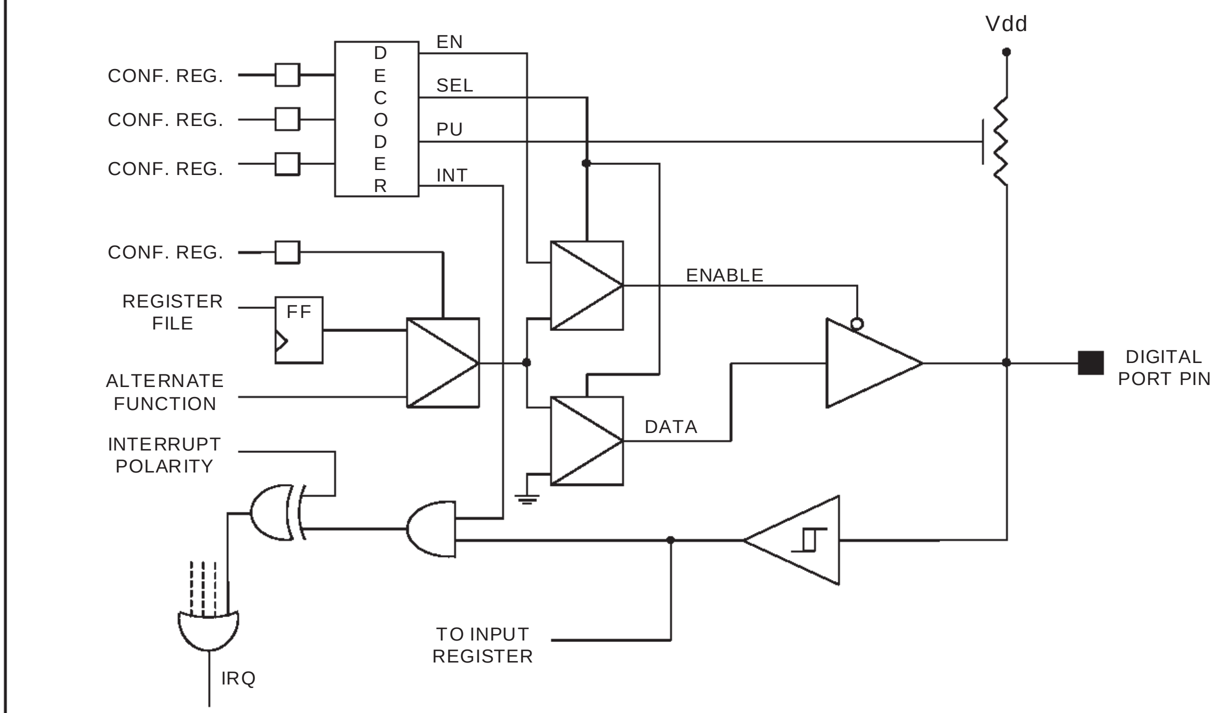
When an on-chip peripheral is configured to use a pin, the correct I/O mode of the related pin should be selected by selecting one of the appropriate modes. See the Registers description in order to obtain the right configurations.Some peripherals, as for example the I^{2}C peripheral, directly drive the pin configuration according to the current function, overriding the user configuration.

7.6 Register Description

In order to configure the Port's pins, the three Configuration Registers PORT_x_PULLUP, PORT_x_OR and PORT_x_DDR must be configured. The combination of these three registers determine the pin's configuration, according to the scheme shown in Table 7.1.

In order to select between the digital functions or Alternate functions PORT_x_AF register must be configured. Each bit of the configuration registers configures the pin of the corresponding position (example: PORT_A_DDR bit 5 configures the pin PA5).

Figure 7.3 Port Pin Architecture



7.6.1 Configuration Registers.

Port A Pull-Up Register (PORT_A_PULLUP)
Configuration Register 24 (018h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
PUA7	PUA6	PUA5	PUA4	PUA3	PUA2	PUA1	PUA0

Bit 7-0: **PUA7-6** Port A pull-up (see Table 7.1)
0: Port A pin without pull-up
1: Port A pin with pull-up

Port A Option Register (PORT_A_OR)
Configuration Register 25 (019h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
ORA7	ORA6	ORA5	ORA4	ORA3	ORA2	ORA1	ORA0

Bit 7-0: **ORA7-6** Port A option (see Table 7.1)

Port A Data Direction Register (PORT_A_DDR)
Configuration Register 26 (01Ah) Read/Write
Reset Value: 0000 0000 (00h)

7							0
DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0

Bit 7-0: **DDRA7-6** Port A direction (see Table 7.1)
0: Port A pin configured as input
1: Port A pin configured as output

Port A Alternate Fuction (PORT_A_AF)
Configuration Register 27 (01Bh) Read/Write
Reset Value: 0000 0000 (00h)

7							0
AFA7	AFA6	AFA5	AFA4	AFA3	AFA2	AFA1	AFA0

Bit 7: **AFA7** Alternate Function PA7
0: Digital I/O
1: INT

Bit 6: **AFA6** Alternate Function PA6
0: Digital I/O
1: T0OUT

Bit 5: **AFA5** Alternate Function PA5
0: Digital I/O
1: TCLK

Bit 4: **AFA4** Alternate Function PA4
0: Digital I/O
1: TSTRT

Bit 3: **AFA3** Alternate Function PA3
0: Digital I/O
1: TRES

Bit 2: **AFA2** Alternate Function PA2
0: Digital I/O
1: T1OUT

Bit 1: **AFA1** Alternate Function PA1
0: Digital I/O
1: SDA

Bit 0: **AFA0** Alternate Function PA0
0: Digital I/O
1: SCL

Table 7.1 Pin mode configuration

MODE	PU	OR	DDR
Input high impedance	0	0	0
Input with pull-up	1	0	0
Interrupt without pull-up	0	1	0
Interrupt with pull-up	1	1	0
Output push-pull	0	0	1
Output with pull-up	1	0	1
Output open drain	0	1	1
Output weak pull-up	1	1	1

Port B Pull-Up Register (PORT_B_PULLUP)
Configuration Register 28 (01Ch) Read/Write
Reset Value: 0000 0000 (00h)

7							0
PUB7*	PUB6*	PUB5**	PUB4**	PUB3**	PUB2**	PUB1	PUB0

(*) Not used in 20 pin package devices
(**) Not used in 16 pin package devices

Bit 7-0: **PUB7-6** Port B pull-up (see Table 7.1)
0: Port B pin without pull-up
1: Port B pin with pull-up

Port B Option Register (PORT_B_OR)
Configuration Register 29 (01Dh) Read/Write
Reset Value: 0000 0000 (00h)

7							0
ORB7*	ORB6*	ORB5**	ORB4**	ORB3**	ORB2**	ORB1	ORB0

(*) Not used in 20 pin package devices
(**) Not used in 16 pin package devices

Bit 7-0: **ORB7-6** Port B option (see Table 7.1)

Port B Data Direction Register (PORT_B_DDR)
Configuration Register 30 (01Eh) Read/Write
Reset Value: 0000 0000 (00h)

7							0
DDRB7*	DDRB6*	DDRB5**	DDRB4**	DDRB3**	DDRB2**	DDRB1	DDRB0

(*) Not used in 20 pin package devices
(**) Not used in 16 pin package devices

Bit 7-0: **DDRB7-6** Port B direction (see Table 7.1)
0: Port B pin configured as input
1: Port B pin configured as output

Port B Alternate Fuction (PORT_B_AF)
Configuration Register 31 (01Fh) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	AFB3	AFB2	AFB1	AFB0

Note: *This register is not used in 16 pin devices*

Bit 7-4: Not Used

Bit 3: **AFB3** Alternate Function PB3
0: Digital I/O
1: SS

Bit 2: **AFB2** Alternate Function PB2
0: Digital I/O
1: SDI

Bit 1: **AFB1** Alternate Function PB1
0: Digital I/O
1: SDO

Bit 0: **AFB0** Alternate Function PB0
0: Digital I/O
1: SCK

Port C Pull-Up Register (PORT_C_PULLUP)
Configuration Register 32 (020h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	PUC5	PUC4	PUC3	PUC2	PUC1	PUC0

Note: *This register is not used in 16/20 pin devices*

Bit 7-6: Not Used

Bit 5-0: **PUC7-6** Port C pull-up (see Table 7.1)
0: Port C pin without pull-up
1: Port C pin with pull-up

Port C Option Register (PORT_C_OR)
Configuration Register 33 (021h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	ORC5	ORC4	ORC3	ORC2	ORC1	ORC0

Note: *This register is not used in 16/20 pin devices*

Bit 7-6: Not Used

Bit 5-0: **ORC7-6** Port C option (see Table 7.1)

Port C Data Direction Register (PORT_C_DDR)
Configuration Register 34 (022h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0

Note: *This register is not used in 16/20 pin devices*

Bit 7-6: Not Used

Bit 5-0: **DDRC7-6** Port C direction (see Table 7.1)
0: Port C pin configured as input
1: Port C pin configured as output

Note: *in order to achieve low current consuption, the port pins must be configured as input pull-up, even though they are not existing in the package. For example in 20 pin devices, the pins PB6-7 and PC0-7 must be configured in input pull-up.*

7.6.2 Input Registers.

Port A Data Input Register (PORT_A_IN)
Input Register 0 (00h) Read only
Reset Value: XXXX XXXX

7							0
PAI7	PAI6	PAI5	PAI4	PAI3	PAI2	PAI1	PAI0

Bit 7-0: **PAI7-0** Port A Input data

The logical level applied in the Port A pins, configured as digital input, can be achieved by reading this register.

Port B Data Input Register (PORT_B_IN)
Input Register 1 (01h) Read only
Reset Value: XXXX XXXX

7							0
PBI7*	PBI6*	PBI5**	PBI4**	PBI3**	PBI2**	PBI1	PBI0

(*) Not used in 20 pin package devices
(**) Not used in 16 pin package devices

Bit 7-0: **PBI7-0** Port B Input data

The logical level applied in the Port B pins, configured as digital input, can be achieved by reading this register.

Port C Data Input Register (PORT_C_IN)
Input Register 2 (02h) Read only
Reset Value: XXXX XXXX

7							0
-	-	PCI5	PCI4	PCI3	PCI2	PCI1	PCI0

Note: *This register is not used in 16/20 pin devices*

Bit 7-6: Not Used

Bit 5-0: **PCI5-0** Port C Input data

The logical level applied in the Port C pins, configured as digital input, can be achieved by reading this register.

7.6.3 Output Registers.

Port A Data Output Register (PORT_A_OUT)
Output Register 0 (00h) Write only
Reset Value: 0000 0000 (00h)

7							0
PAO7	PAO6	PAO5	PAO4	PAO3	PAO2	PAO1	PAO0

Bit 7-0: **PAO7-0** Port A Output data

The logical values written in these register bits are put in the Port A pins configured as digital output.

Port B Data Output Register (PORT_B_OUT)
Output Register 1 (01h) Write only
Reset Value: 0000 0000 (00h)

7							0
PBO7*	PBO6*	PBO5**	PBO4**	PBO3**	PBO2**	PBO1	PBO0

(*) Not used in 20 pin package devices
(**) Not used in 16 pin package devices

Bit 7-0: **PBO7-0** Port B Input data

The logical values written in these register bits are put in the Port B pins configured as digital output.

Port C Data Output Register (PORT_C_OUT)

Output Register 2 (02h) Write only

Reset Value: 0000 0000 (00h)

7							0
-	-	PCO5	PCO4	PCO3	PCO2	PCO1	PCO0

Note: *This register is not used in 16/20 pin devices*

Bit 7-6: Not Used

Bit 5-0: **PCO7-0** Port C Input data

The logical values written in these register bits are put in the Port C pins configured as digital output.

8 FUZZY COMPUTATION (DP)

The ST52F500/F503/F504 Decision Processor (DP) main features are:

- Up to 8 Inputs with 8-bit resolution;
- 1 Kbyte of Program/Data Memory available to store more than 300 to Membership Functions (Mbfs) for each Input;
- Up to 128 Outputs with 8-bit resolution;
- Possibility of processing fuzzy rules with an UNLIMITED number of antecedents;
- UNLIMITED number of Rules and Fuzzy Blocks.

The limits on the number of Fuzzy Rules and Fuzzy program blocks are only related to the Program/Data Memory size.

8.1 Fuzzy Inference

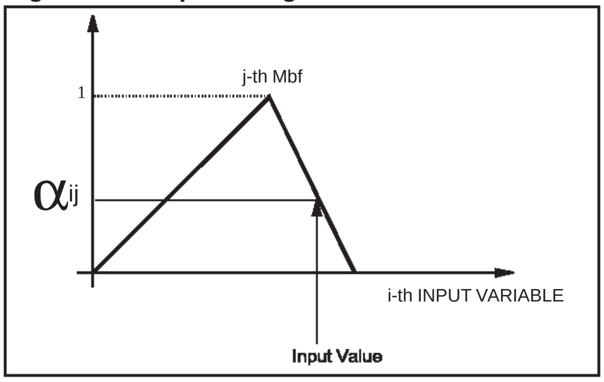
The block diagram shown in Figure 8.1 describes the different steps performed during a Fuzzy algorithm. The ST52F500/F503/F504 Core allows for the implementation of a Mamdami type fuzzy inference with crisp consequents. Inputs for fuzzy inference are stored in 8 dedicated Fuzzy input registers. The LDFR instruction is used to set the Input Fuzzy registers with values stored in the Register File. The result of a Fuzzy inference is stored directly in a location of the Register File.

8.2 Fuzzyfication Phase

In this phase the intersection (alpha weight) between the input values and the related Mbfs (Figure 8.2) is performed.

Eight Fuzzy Input registers are available for Fuzzy inferences.

Figure 8.2 Alpha Weight Calculation



After loading the input values by using the LDFR assembler instruction, the user can start the fuzzy inference by using the FUZZY assembler instruction. During fuzzyfication: input data is transformed in the activation level (alpha weight) of the Mbf's.

8.3 Inference Phase

The Inference Phase manages the alpha weights obtained during the fuzzyfication phase to compute the truth value (ω) for each rule.

This is a calculation of the maximum (for the OR operator) and/or minimum (for the AND operator) performed on alpha values according to the logical connectives of Fuzzy Rules.

Several conditions may be linked together by linguistic connectives AND/OR, NOT operators and brackets.

The truth value ω and the related output singleton are used by the Defuzzyfication phase, in order to complete the inference calculation.

Figure 8.1 Fuzzy Inference

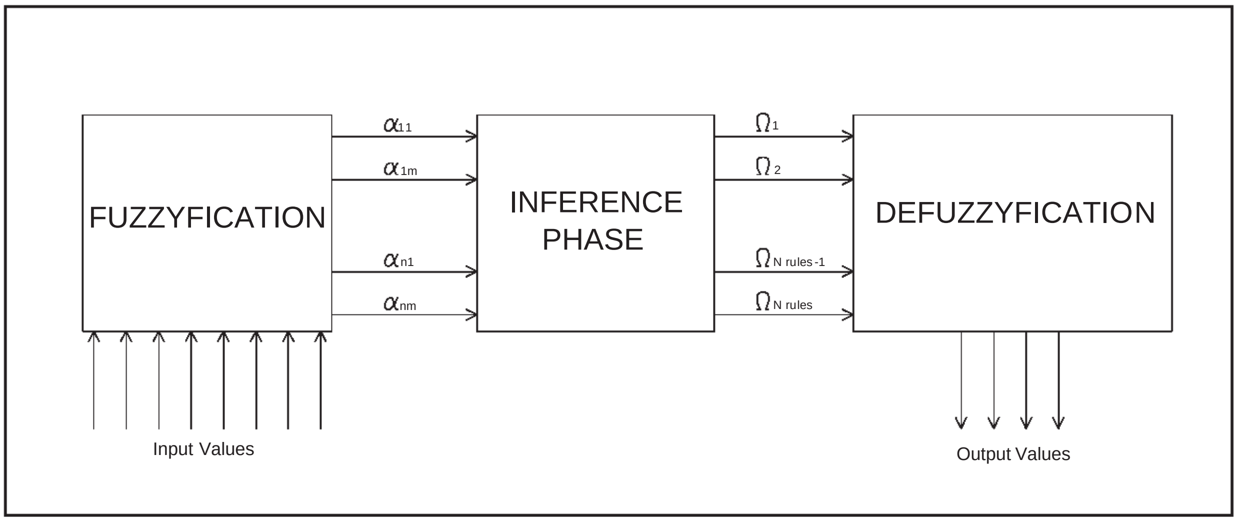
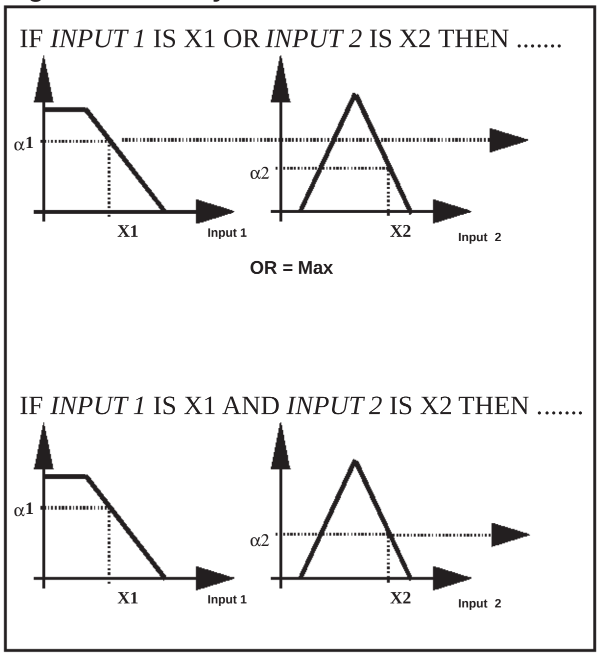


Figure 8.3 Fuzzyfication


IF *INPUT 1* IS X1 OR *INPUT 2* IS X2 THEN ....
α1
α2
X1
Input 1
X2
Input 2
OR = Max
IF *INPUT 1* IS X1 AND *INPUT 2* IS X2 THEN ....
α1
α2
X1
Input 1
X2
Input 2

8.4 Defuzzyfication

In this phase the output crisp values are determined by implementing the consequent part of the rules.

Each consequent Singleton X_i is multiplied by its weight values ω_i, calculated by the Decision processor, in order to compute the upper part of the Defuzzyfication formula.

Each output value is obtained from the consequent crisp values (X_j) by carrying out the following Defuzzyfication formula:

$$Y_i = \frac{\sum_{j}^{N} X_{ij}\omega_{ij}}{\sum_{j}^{N} \omega_{ij}}$$

where:

i = identifies the current output variable

N = number of the active rules on the current output

ω_{ij} = weight of the j-th singleton

X_{ij} = abscissa of the j-th singleton

The Decision Processor outputs are stored in the RAM location i-th specified in the assembler instruction OUT i.

8.5 Input Membership Function

The Decision Processor allows the management of triangular Mbfs. In order to define an Mbf, three different parameters must be stored on the Program/Data Memory (see Figure 8.4):

- the vertex of the Mbf: **V**;
- the length of the left semi-base: **LVD**;
- the length of the right semi-base: **RVD**;

In order to reduce the size of the memory area and the computational effort the vertical range of the vertex is fixed between 0 and 15 (4 bits)

By using the previous memorization method different kinds of triangular Membership Functions may be stored. Figure 8.5 shows some examples of valid Mbfs that can be defined in ST52F500/F503/F504.

Each Mbf is then defined storing 3 bytes in the first Kbyte of the Program/Data Memory.

The Mbf is stored by using the following instruction:

MBF *n_mbf lvd v rvd*

where:

n_mbf is a tag number that identifies the Mbf

lvd, *v*, and *rvd* are the parameters that describe the Mbf's shape as described above.

Figure 8.4 Mbfs Parameters

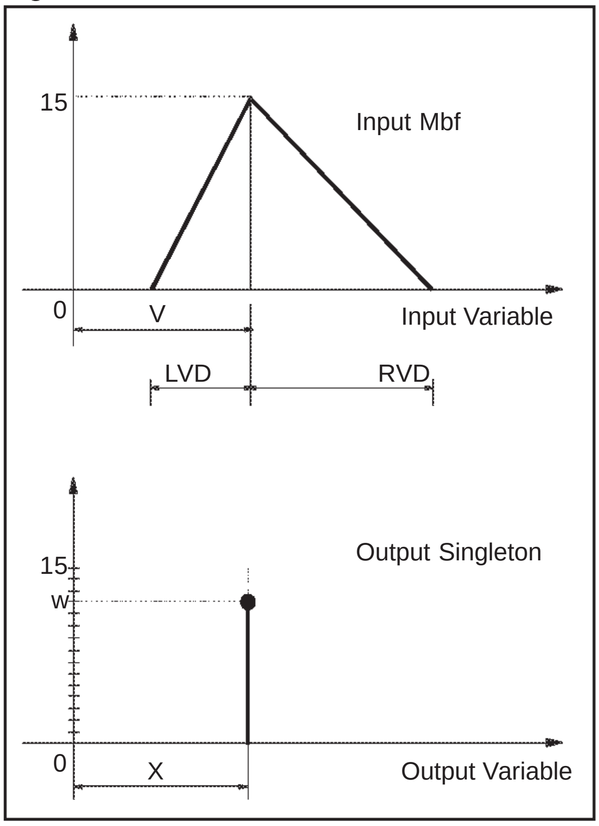


Figure 8.5 Example of valid Mbfs

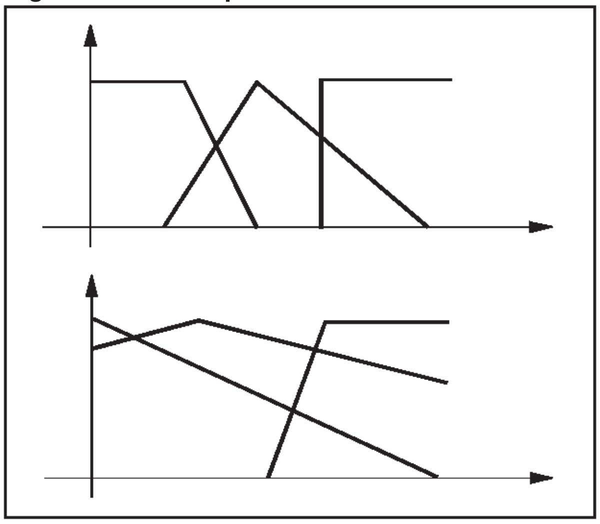
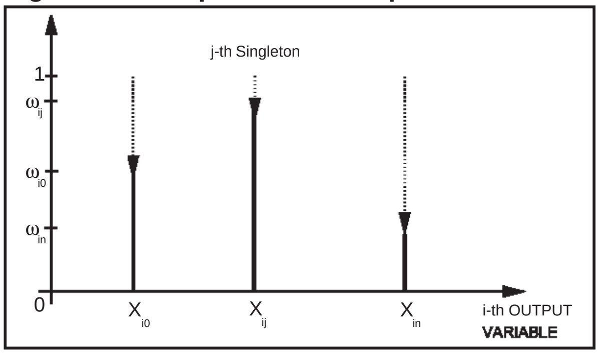


Figure 8.6 Output Membership Functions



8.6 Output Singleton

The Decision Processor uses a particular kind of membership function called Singleton for its output variables. A Singleton doesn't have a shape, like a traditional Mbf, and is characterized by a single point identified by the couple (X, w), where w is calculated by the Inference Unit as described earlier. Often, a Singleton is simply identified with its Crisp Value X.

8.7 Fuzzy Rules

Rules can have the following structures:

if A op B op C...........then Z

if (A op B) op (C op D op E...) ....then Z

where *op* is one of the possible linguistic operators (AND/OR)

In the first case the rule operators are managed sequentially; in the second one, the priority of the operator is fixed by the brackets.

Each rule is codified by using an instruction set, the inference time for a rule with 4 antecedents and 1 consequent is about 3 microseconds at 20 MHz.

The Assembler Instruction Set used to manage the Fuzzy operations is reported in the table below.

Table 8.1 Fuzzy Instructions Set

Instruction	Description
MBF *n_mbf lvd v rvd*	Stores the Mbf *n_mbf* with the shape identified by the parameters *lvd*, *v* and *rvd*
IS *n m*	Fixes the alpha value of the input n with the Mbf m
ISNOT *n m*	Calculates the complementary alpha value of the input n with the Mbf m.
FZAND	Implements the Fuzzy operation AND
FZOR	Implements the Fuzzy operation OR
CON *crisp*	Multiplies the crisp value with the last ω weight
OUT *n_out*	Performs Defuzzyfication and stores the currently Fuzzy output in the register *n_out*
FUZZY	Starts the computation of a sigle fuzzy variable
( )	Modify the priority in the rule evaluation

Example 1:

IF Input$_1$ IS NOT Mbf$_1$ AND Input$_4$ is Mbf$_{12}$ OR Input$_3$ IS Mbf$_8$ THEN Crisp$_1$

is codified by the following instructions:

ISNOT 1 1	calculates the NOT α value of Input$_1$ with Mbf$_1$ and stores the result in internal registers
FZAND	implements the operation AND between the previous and the next alpha value evaluated
IS 4 12	fixes the α value of Input$_4$ with Mbf$_{12}$ and stores the result in internal registers
FZOR	implements the operation OR between the previous and the next alpha value evaluated
IS 3 8	fixes the α value of Input$_3$ with Mbf$_8$ and stores the result in internal registers
CON *crisp$_1$*	multiplies the result of the last Ω operation with the crisp value *crisp$_1$*

Example 2, the priority of the operator is fixed by the brackets:

IF (Input$_3$ IS Mbf$_1$ AND Input$_4$ IS NOT Mbf$_{15}$) OR (Input$_1$ IS Mbf$_6$ OR Input$_6$ IS NOT Mbf$_{14}$) THEN Crisp$_2$

(	parenthesis open to change the priority
IS 3 1	fixes the α value of Input$_3$ with Mbf$_1$ and stores the result in internal registers
FZAND	implements the operation AND between the previous and the next alpha value evaluated
ISNOT 4 15	calculates the NOT α value of Input$_4$ with Mbf$_{15}$ and stores the result in internal registers
)	parenthesis closed
FZOR	implements the operation OR between the previous and the next alpha value evaluated
(	parenthesis open to change the priority
IS 1 6	fixes the α value of Input$_1$ with Mbf$_6$ and stores the result in internal registers
FZOR	implements the operation OR between the previous and the next alpha value evaluated
ISNOT 2 14	calculates the NOT α value of Input$_6$ with Mbf$_{14}$ and stores the result in internal registers
)	parenthesis closed
CON *crisp$_2$*	multiplies the result of the last Ω operation with the crisp value *crisp$_2$*

At the end of the fuzzy rules related to the current Fuzzy Variable, by using the instruction **OUT *reg***, the specified register is written with the computed value. Afterwards, the control of the algorithm returns to the CU. The next Fuzzy Variable evaluation must start again with a FUZZY instruction.

9 INSTRUCTION SET

ST52F500/F503/F504 supplies 107 (98 + 9 Fuzzy) instructions that perform computations and control the device. Computational time required for each instruction consists of one clock pulse for each Cycle plus 2 clock pulses for the decoding phase. Total computation time for each instruction is reported in Table 9.1

The ALU of ST52F500/F503/F504 can perform multiplication (MULT) and division (DIV). Multiplication is performed by using 8 bit operands storing the result in 2 registers (16 bit values), see Figure 2.3.

Division is performed between a 16 bit dividend and an 8 bit divider, the result and the remainder are stored in two 8-bit registers (see Figure 2.4).

9.1 Addressing Modes

ST52F500/F503/F504 instructions allow the following addressing modes:

- Inherent: this instruction type does not require an operand because the opcode specifies all the information necessary to carry out the instruction. Examples: NOP, SCF.
- Immediate: these instructions have an operand as a source immediate value. Examples: LDRC, ADDI.
- Direct: the operands of these instructions are specified with the direct addresses. The operands can refer (according to the opcode) to addresses belonging to the different addressing spaces. Example: SUB, LDRE.
- Indirect: data addresses that are required are found in the locations specified as operands. Both source and/or destination operands can be addressed indirectly. The operands can refer, (according to the opcode) to addresses belonging to different addressing spaces.
  Examples: LDRR(reg1),(reg2);
  LDER mem_addr,(reg1).
- Bit Direct: operands of these instructions directly address the bits of the specified Register File locations. Examples: BSET, BTEST.

9.2 Instruction Types

ST52F500/F503/F504 supplies the following instruction types:

- Load Instructions
- Arithmetic and Logic Instructions
- Bitwise instructions
- Jump Instructions
- Interrupt Management Instructions
- Control Instructions

The instructions are listed in Table 9.1

Table 9.1 Instruction Set

Load Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
BLKSET	BLKSET const	2	(*)	-	-	-
GETPG	GETPG regx	2	7	-	-	-
LDCE	LDCE confx,memy	3	8/9	-	-	-
LDCI	LDCI confx, const	3	7	-	-	-
LDCNF	LDCNF regx, conf	3	7	-	-	-
LDCR	LDCR confx, regy	3	8	-	-	-
LDER	LDER memx, regy	3	10	-	-	-
LDER	LDER (regx),(regy)	3	11	-	-	-
LDER	LDER (regx), regy	3	10	-	-	-
LDER	LDER memx,(regy)	3	11	-	-	-
LDFR	LDFR fuzzyx, regy	3	8	-	-	-

Load Instructions (continued)

LDPE	LDPE outx, memy	3	8/9	-	-	-
LDPE	LDPE outx, (regy)	3	9/10	-	-	-
LDPI	LDPI outx, const	3	7	-	-	-
LDPR	LDPR outx, regy	3	8	-	-	-
LDRC	LDRC regx, const	3	7	-	-	-
LDRE	LDRE regx, memy	3	8/9	-	-	-
LDRE	LDRE (regx), (regy)	3	10/11	-	-	-
LDRE	LDRE (regx), memy	3	9/10	-	-	-
LDRE	LDRE regx, (regy)	3	9/10	-	-	-
LDRI	LDRI regx, inpx	3	7	-	-	-
LDRR	LDRR regx, regy	3	9	-	-	-
LDRR	LDRR (regx), (regy)	3	10	-	-	-
LDRR	LDRR (regx), regy	3	9	-	-	-
LDRR	LDRR regx, (regy)	3	10	-	-	-
PGSET	PGSET const	2	4	-	-	-
PGSETR	PGSETR regx	2	5	-	-	-
POP	POP regx	2	7	-	-	-
PUSH	PUSH regx	2	8	-	-	-

Arithmetic Instructions

Mnemonic	Instruction	Bytes	Cycles	Z	S	C
ADD	ADD regx, regy	3	9	I	-	I
ADDC	ADDC regx, regy	3	9	I	-	I
ADDI	ADDI regx, const	3	8	I	-	I
ADDIC	ADDIC regx, const	3	8	I	-	I
ADDO	ADDO regx, regy	3	11	I	I	I
ADDOC	ADDOC regx, regy	3	11	I	I	I
ADDOI	ADDOI regx, const	3	10	I	I	I
ADDOIC	ADDOICregx,cons	3	10	I	I	I
AND	AND regx, regy	3	9	I	-	-
ANDI	ANDI regx,const	3	8	I	-	-
CP	CP regx, regy	3	8	I	I	-
CPI	CPI regx,const	3	7	I	I	-
DEC	DEC regx	2	7	I	I	-

Arithmetic Instructions (continued)

DIV	DIV regx, regy	3	16	I	I	I
INC	INC regx	2	7	I	-	I
MIRROR	MIRROR regx	2	7	I	-	-
MULT	MULT regx, regy	3	11	I	-	-
NOT	NOT regx	2	7	I	-	-
OR	OR regx, regy	3	9	I	-	-
ORI	ORI regx, const	3	8	I	-	-
SUB	SUB regx, regy	3	9	I	I	-
SUBI	SUBI regx, const	3	8	I	I	-
SUBIS	SUBIS regx, const	3	8	I	I	-
SUBO	SUBO regx, regy	3	11	I	I	I
SUBOI	SUBOI regx,	3	10	I	I	I
SUBOIS	SUBOISregx,const	3	10	I	I	I
SUBOS	SUBOS regx, regy	3	11	I	I	I
SUBS	SUBS regx, regy	3	9	I	I	-
RCF	RCF	1	4	-	-	I
RSF	RSF	1	4	-	I	-
RZF	RZF	1	4	I	-	-
SCF	SCF	1	4	-	-	I
SSF	SSF	1	4	-	I	-
SZF	SZF	1	4	I	-	-
XOR	XOR regx, regy	3	9	I	-	-
XORI	XORI regx, cons	3	8	I	-	-

Bitwise Instructions

Mnemonic	Instruction	Bytes	Cycles	Z	S	C
ASL	ASL regx	2	7	I	-	I
ASR	ASR regx	2	7	I	I	-
BNOT	BNOT regx, bit	3	8	I	-	-
BRES	BRES regx, bit	3	8	I	-	-
BSET	BSET regx, bit	3	8	I	-	-
BTEST	BTEST regx, bit	3	7	I	-	-
MTEST	MTEST regx,	3	7	I	-	-
RLC	RLC regx	2	7	I	-	I

Bitwise Instructions (continued)						
ROL	ROL regx	2	7	I	-	I
ROR	ROR regx	2	7	I	I	-
RRS	RRS regx	2	7	I	I	-

Jump Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
CALL	CALL addr	3	11	-	-	-
JP	JP addr	3	6	-	-	-
JPC	JPC addr	3	5/6	-	-	-
JPNC	JPNC addr	3	5/6	-	-	-
JPNS	JPNS addr	3	5/6	-	-	-
JPNZ	JPNZ addr	3	5/6	-	-	-
JPS	JPS addr	3	5/6	-	-	-
JPZ	JPZ addr	3	5/6	-	-	-
RET	RET	1	8	-	-	-

Interrupt Management Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
HALT	HALT	1	4/13	-	-	-
MEGI	MEGI	1	6/11	-	-	-
MDGI	MDGI	1	5	-	-	-
RETI	RETI	1	9	-	-	-
RINT	RINT INT	2	6	-	-	-
UDGI	UDGI	1	5	-	-	-
UEGI	UEGI	1	6/11	-	-	-
TRAP	TRAP	1	9	-	-	-
WAITI	WAITI	1	7/10	-	-	-

Control Instructions						
Mnemonic	**Instruction**	**Bytes**	**Cycles**	**Z**	**S**	**C**
FUZZY	FUZZY	1	4	-	-	-
NOP	NOP	1	5	-	-	-
WDTRFR	WDTRFR	1	6	-	-	-
WDTSLP	WDTSLP	1	5	-	-	-

Notes:

regx, regy:	Register File Address
memx, memy:	Program/Data Memory Addresses
confx, confy:	Configuration Registers Addresses
outx:	Output Registers Addresses
inpx:	Input Registers Addresses
const:	Constant value
fuzzyx:	Fuzzy Input Registers
I	flag affected
-	flag not affected

(*) The instruction BLKSET determines the start of a 32 byte block writing in Flash or EEPROM Program/ Data Memory. During this phase (about 4 ms), the CPU is stopped to executing program instructions. The duration of the BLKSET instruction can be identified with this time.

10 WATCHDOG TIMER

10.1 Functional Description

The Watchdog Timer (WDT) is used to detect the occurrence of a software fault, usually generated by external interference or by unforeseen logical conditions, which causes the application program to abandon its normal sequence. The WDT circuit generates an ICU reset on expiry of a programmed time period, unless the program refreshes the WDT before the end of the programmed time delay. Sixteen different delays can be selected by using the WDT configuration register.

After the end of the delay programmed by the configuration register, if the WDT is active, it starts a reset cycle pulling the reset signal low.

Once the WDT is activated, the application program has to refresh the counter (by the WDTSFR instruction) during normal operation in order to prevent an ICU reset.

In ST52F500/F503/F504 devices it is possible to choose between “Hardware” or “Software” Watchdog. The Hardware WDT allows the counting to avoid unwanted stops for external interferences. The first mode is always enabled unless the Option Byte 4 (WDT_EN) is written with a special code (10101010b): only this code can switch the WDT in “Software” Mode, the other 255 possibilities keep the “Hardware” Mode enabled.

When the software mode is enabled, it is possible to stop the WDT during the user program executions by using the WDTSLP instruction.

When the WDT is in Hardware Mode, neither the WDTSLP instruction nor external interference can stop the counting. The “Hardware” WDT is always enabled after a Reset.

The working frequency of WDT (PRES CLK in the Figure 10.1) is equal to the clock master. The clock master is divided by 500, obtaining the WDT CLK signal that is used to fix the timeout of the WDT.

According to the WDT_CR Configuration Register values, a WDT delay between 0.1ms and 937.5ms can be defined when the clock master is 5 MHz. By changing the clock master frequency the timeout delay can be calculated according to the configuration register values. The first 4 bits of the WDT_CR register are used, obtaining 16 different delays.

Table 10.1 Watchdog Timing Range (5 MHz)

	WDT timeout period (ms)
min	0.1
max	937.5

10.2 Register Description

SW Watchdog Enable (WDT_EN)

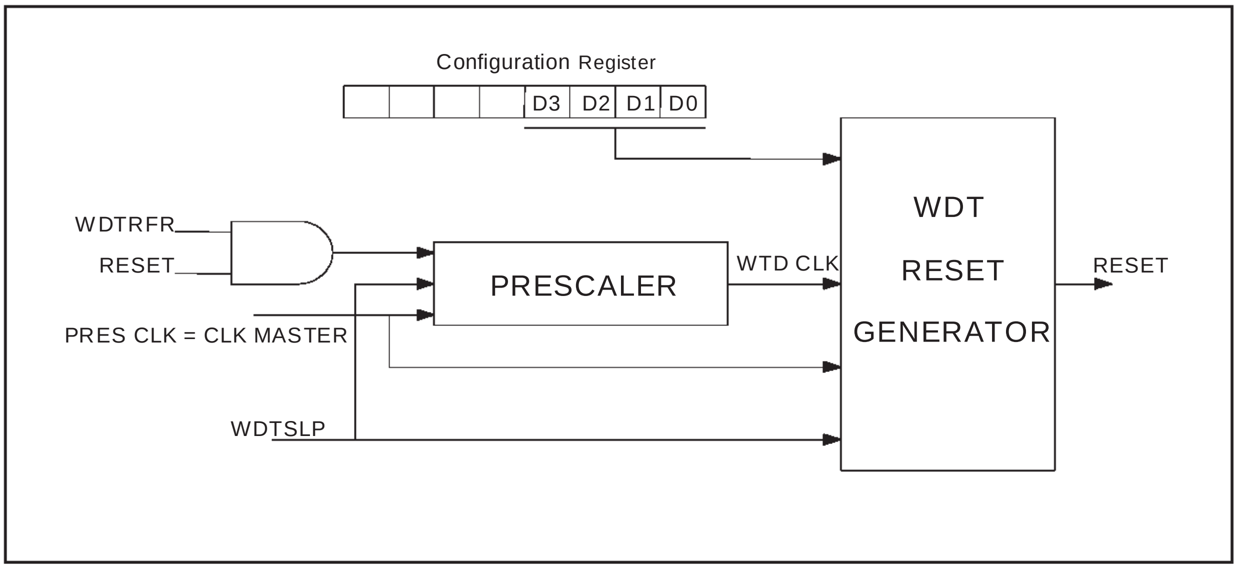
Option Byte 4 (04h)

Reset Value: 0000 0000 (00h)

7							0
WDTEN7	WDTEN6	WDTEN5	WDTEN4	WDTEN3	WDTEN2	WDTEN1	WDTEN0

Bit 7-0: **WDTEN7-0** SW Watchdog Enable byte
Writing the code 10101010 in this byte the Software Watchdog mode is enabled.

Figure 10.1 Watchdog Block Diagram



Watchdog Control Register (WDT_CR)

Configuration Register 7 (07h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	D3	D2	D1	D0

Bit 7-4: Not Used

Bit 3-0: **D3-0** Watchdog Clock divisor factor bits

The Watchdog Clock (WDT CLK) is divided by the numeric factor determined by these bits, according with Table 10.2 and the following formula:

$$Timeout(ms) = \frac{5 \times 10^5 \times DivisionFactor}{Clock(MHz)}$$

Table 10.2 Watchdog Timeout configuration examples

WDT_CR(3:0)	Division Factor	Timeout Values (ms)		
		5 MHz	**10 MHz**	**20MHz**
0000	1	0.1	0.05	0.025
0001	625	62.5	31.25	15.625
0010	1250	125	62.5	31.25
0011	1875	187.5	93.75	46.875
0100	2500	250	125	62.5
0101	3125	312.5	156.25	78.125
0110	3750	375	187.5	93.75
0111	4375	437.5	218.75	109.375
1000	5000	500	250	125
1001	5625	562.5	281.25	140.625
1010	6250	625	312.5	156.25
1011	6875	687.5	343.75	171.875
1100	7500	750	375	187.5
1101	8125	812.5	406.25	203.125
1110	8750	875	437.5	218.75
1111	9375	937.5	468.75	234.375

11 PWM/TIMERS

11.1 Introduction

ST52x520/521 offers two on-chip PWM/Timer peripherals. All ST52x520/521 PWM/Timers have the same internal structure. The timer consists of a 16-bit counter with a 16-bit programmable Prescaler, giving a maximum count of 2^{32} (see Figure 11.1).

Each timer has two different working modes, which can be selected by setting the correspondent bit TxMOD of the PWMx_CR1 Configuration Register: Timer Mode and PWM (Pulse Width Modulation) Mode.

All the Timers have Autoreload Functions; in PWM Mode the reload value can be set by the user.

Each timer output is available on the apposite external pins configured in Alternate Function and in one of the Output modes.

PWM/Timer 0 can also use external START/STOP signals in order to perform Input capture and Output compare, external RESET signal, and external CLOCK to count external events: T0STRT, T0RES and T0CLK pins. In addition, the START/STOP and RESET signals have configurable polarity (falling or rising edge).

Remark: *To use T0RST, T0STR, T0CLK external signals the related pins must be configured in Alternate Function and in one of Input modes.*

For each timer, the contents of the 16-bit counter are incremented on the Rising Edge of the 16-bit prescaler output (PRESCOUT) and it can be read at any instant of the counting phase by accessing the Input Registers PWMx_COUNT_IN_x; the value is stored in two 8-bit registers (MSB and LSB) for each PWM/Timer.

The Input Registers couple PWMx_CAPTURE_x store the counter value after the last Stop signal (only Timer Mode). The counter value is not stored after a Reset Signal.

The peripheral status can also be read from the Input Registers (one for each Timer). These registers report START/STOP, SET/RESET status, TxOUT signal and the counter overflow flag. This last signal is set after the first EOC and it is reset by a Timer RESET (internal or external).

11.2 Timer Mode

Timer Mode is selected writing 0 in the TxMOD bit.

Each Timer requires three signals: Timer Clock (TMRCLKx), Timer Reset (TxRES) and Timer Start (TxSTRT) (see Figure 11.1). Each of these signals can be generated internally, and/or externally only for Timer 0, by using T0RES, T0STRT and T0CLK pins.

The Prescaler output (PRESCOUT) increments the Counter value on the rising edge. PRESCOUT is obtained from the internal clock signal (CLKM) or, only for TIMER0, from the external signal provided on the apposite pin.

Note: *The external clock signal applied on the T0CLK pin must have a frequency that is at least two times smaller than the internal master clock.*

The prescaler output period can be selected by setting the TxPRESC bits with one of the 17 division factors available. TMRCLK frequency is divided by a factor equal to the power of two of the prescaler values (up to 2^{16}).

TxRES resets the content of the 16-bit counter to zero. It is generated by writing 0 in the TxRES bit of the PWMx_CR1 Configuration Register and/or it can be driven by the T0RES pin if configured (only Timer0).

Figure 11.1 PWM/Timer Counter block diagram

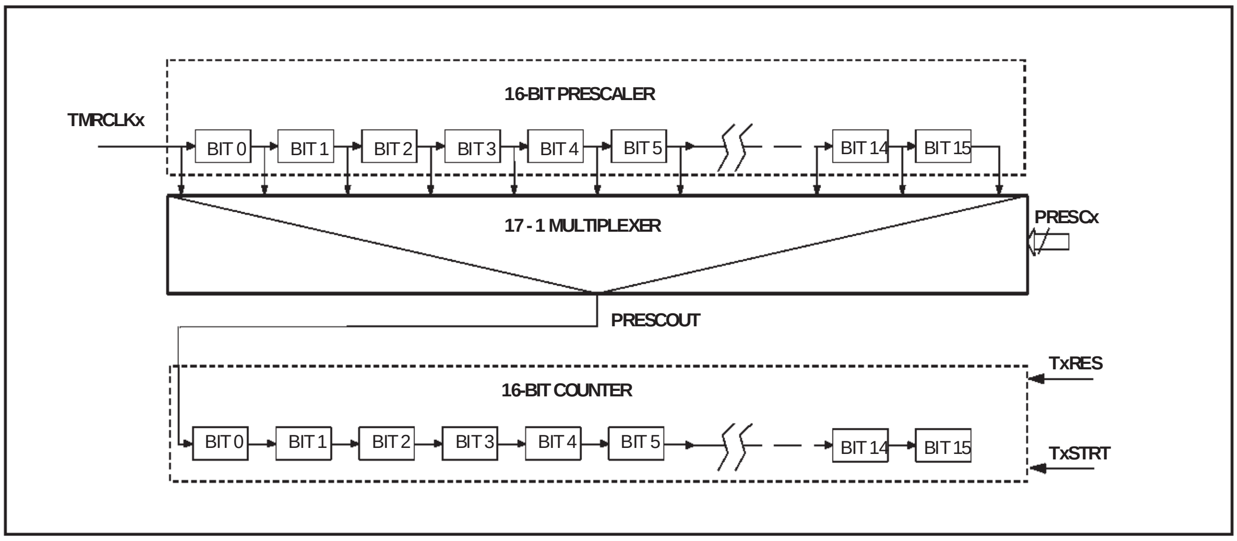
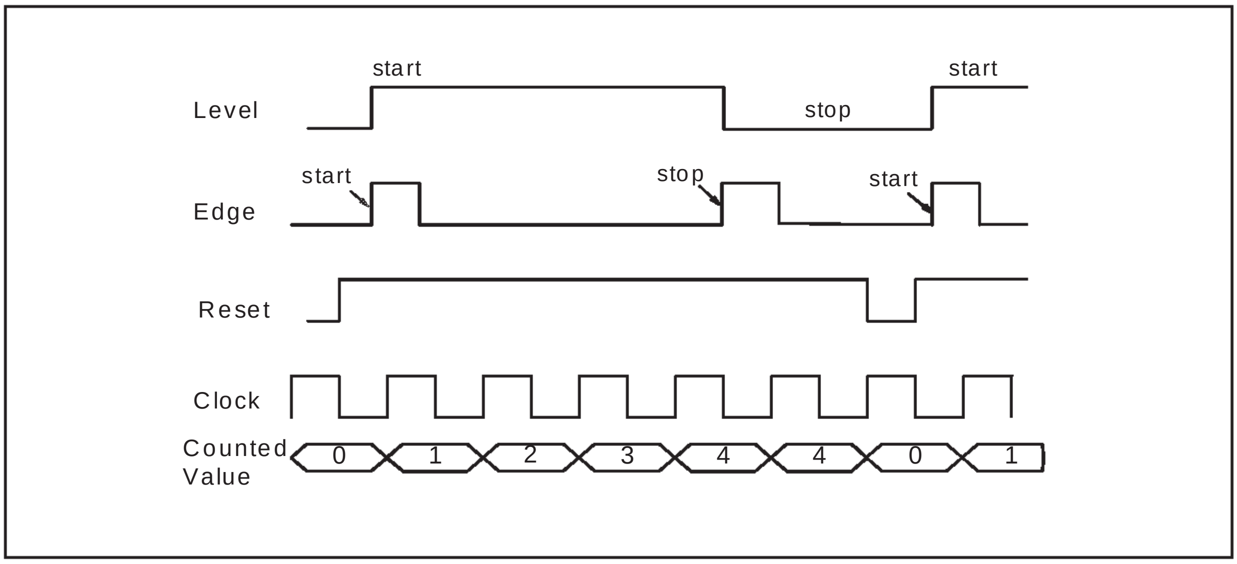


Figure 11.2 Timer 0 External Start/Stop Mode



TxSTRT signal starts/stops the Timer from counting only if the peripherals are configured in Timer mode. The Timers are started by writing 1 in the TXSTRT bit of the PWMx_CR1 and are stopped by writing 0. This signal can be generated internally and/or externally by forcing the T0STRT pin (only TIMER0).

TIMER 0 START/STOP can be given externally on the T0STRT pin. In this case, the T0STRT signal allows the user to work in two different configurable modes:

- LEVEL (Time Counter): If the T0STRT signal is high, the Timer starts counting. When the T0STRT is low the timer stops counting and the 16-bit current value is stored in the PWM0_COUNT_IN_x Input Registers couple.
- EDGE (Period Counter): After reset, on the first T0STRT rising edge, TIMER 0 starts counting and at the next rising edge it stops. In this manner the period of an external signal may be measured.

The same modes are available for the T0RES pin signal.

The polarity of the T0SRTR Start/Stop signal can be changed by setting the STRPOL and RESPOL bits in the INT_POL Configuration Register (01h bit 3 and 4). When these bits are set, the PWM/Timer 0 is Started/Set on the low level or in the falling edge of the signal applied in the pins.

The Timer output signal, TxOUT, is a signal with a frequency equal to the one of the 16 bit-Prescaler output signal, PRESCOUTx, divided by a 16-bit counter set by writing the Output Register couple PWMx_COUNT_OUT_x.

Note: *the contents of these registers upgrades the Timer counter after it stops counting. Since the register couple is written in two steps this can cause side effects. In order to avoid this, the user should write the MSB before writing the LSB: actually, the 16-bit value is latched in parallel when the LSB is written. By writing only the LSB (and MSB equal to 0), the PWM/Timer is used as with an 8 bit counter.*

There can be two types of TxOUT waveforms:

- type 1: TxOUT waveform equal to a square wave with a 50% duty-cycle
- type 2: TxOUT waveform equal to a pulse signal with the pulse duration equal to the Prescaler output signal.

Figure 11.3 TxOUT Signal Types

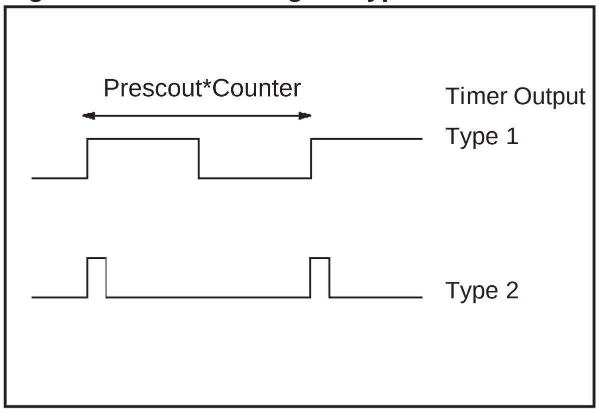
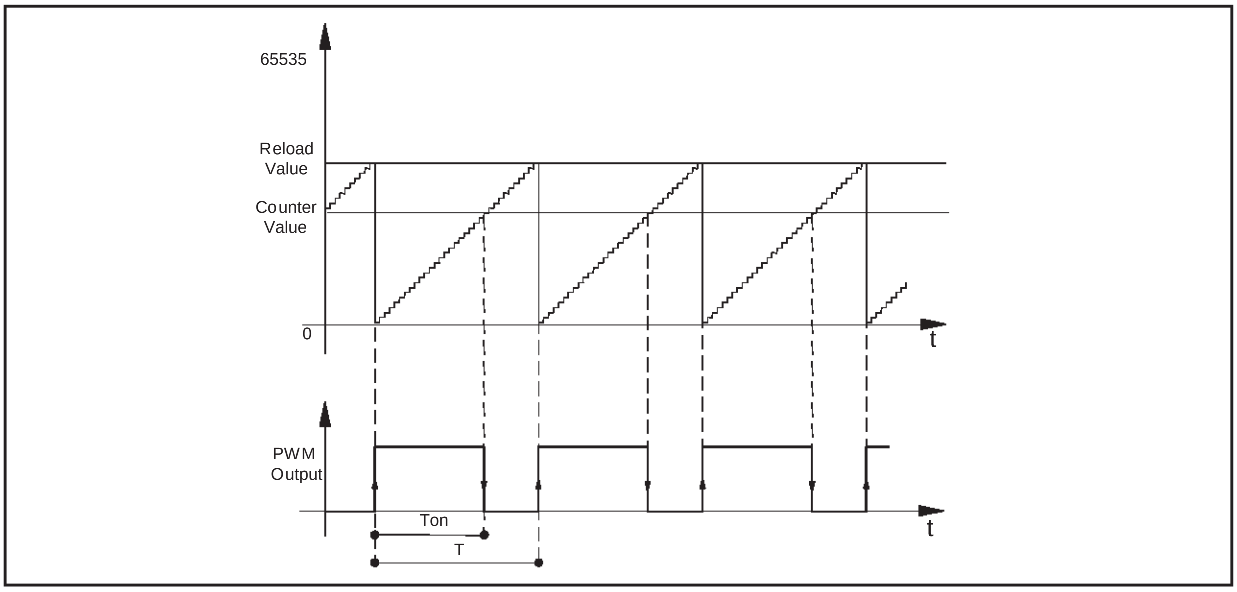


Figure 11.4 PWM Mode with Reload



11.3 PWM Mode

The PWM working mode for each timer is obtained by setting the TxMOD bit of the Configuration Register PWMx_CR1.

The TxOUT signal in PWM Mode consists of a signal with a fixed period, whose duty cycle can be modified by the user.

The TxOUT period is fixed by setting the 16-bit Prescaler bits (TxPRESC) in the PWMx_CR2 and the 16-bit Reload value by writing the relative Output Registers couple PWMx_RELOAD_x. The 16-bit Prescaler divides the master clock CLKM by powers of two, determining the maximum length period.

Reload determines the maximum value that the counter can count before starting a new period. The use of the two 16-bit values allows the TxOUT period to be set with more precision when needed. By setting the Reload value the counting resolution decreases. In order to obtain the maximum resolution, Reload value should be set to 0FFFFh and the period corresponds to the one established by the Prescaler value.

The value set in the 16-bit counter by writing the Counter Output Registers couple, determines the duty-cycle: when count reaches the Counter value the TxOUT signal changes from high to low level.

The period of the PWM signal is obtained by using the following formula:

$T = PWMx\_RELOAD * 2^{TxPRESC} TMRCLKx$

where TxPRES equals the value set in the TxPRESC bits of the PWMx_CR2 Configuration Register and TMRCLKx is the period of the Timer clock that drives the Prescaler.

The duty cycle of the PWM signal is obtained by the following formula:

$$d_{cycle} = \frac{T_{on}}{T} = \frac{PWMxCOUNT}{PWMxRELOAD}$$

***Note:** the PWM_x_COUNT value must be lower than or equal to the PWM_X_RELOAD value. When it is equal, the TxOUT signal is always at high level. If the Output Register PWM_x_COUNT is 0, TxOUT signal is always at a low level.*

By using a 24 MHz clock a PWM frequency that is close to 100 Khz can be obtained.

The TIMER0 clock CLKM can also be supplied with an external signal, applied on the T0CLK pin, which must have a frequency that is at least two times smaller than the internal master clock.

***Note:** the Timers have to complete the previous counting phase before using a new value of the Counter or of the Reload. If the Counter or the Reload value is changed during counting, the new values of the timer Counter and Reload are only used at the end of the previous counting phase. The Counter buffer is written in two steps (one byte per time) and is latched only after the LSB is written. In order to avoid side effects, the user should write the MSB before writing the LSB. By only writing the LSB, the PWM/Timer is used as with a 8 bit counter. The same mechanism is applied to the two bytes of Reload. Nevertheless, it is recommended that the Reload value be written when the Timer is stopped in order to avoid incongruence with the Counter value. The same* *recommendation is made when reading the two bytes of the counter: It is performed in two steps, so if the timer is running, the carry of the LSB to the MSB can cause the wrong 16-bit value reading. A Reload value greater than 1 must always be used.*

When the Timers are in Reset status, or when the device is reset, the TxOUT pins goes in threestate. If these outputs are used to drive external devices, it is recommended that the related pins be left in the default configuration (Input threestate) or change them in this configuration.

In PWM mode the PWM/Timers can only be Set or Reset: Start/Stop signals do not affect the Timers. TxRES resets the content of the 16-bit counter to zero. It is generated by writing 0 in the corresponding TxRES bit of the PWMx_CR1 Configuration Register and/or it can be driven by the T0RES pin if it is configured (only Timer0).

Warning: *In the ST52x520 device, the TxSTRT signal must be kept to 1 when the Timer is in Set state. This can be achieved by writing 1 in the related bit of the Configuration Register*

11.3.1 Simultaneous Start. The PWM/Timers can be started simultaneously when working in PWM mode. The T0SYNC and T1SYNC bits in PWM0_CR3 Configuration Registers mask the reset of each timer; after enabling each single PWM/Timer. They are started by putting off the mask with a single writing in the PWM0_CR3 Register.

Simultaneous start is also possible in Timer mode. The timers start counting simultaneously, but the output pulses are generated according to the modality configured (square or pulse mode).

11.4 Timer Interrupts

The PWM/Timer can be programmed to generate an Interrupt Request, both on the falling and the rising of the TxOUT signal and when there's a STOP signal (external or internal).

By using the TxIES, TxIER and TxIEF bits of the Configuration Registers PWMx_CR1, the interrupt sources can be switched on/off. All the interrupt sources may be activated at the same time: sources can be distinguished by reading the PWMx_STATUS Input Register.

The interrupt on the falling edge corresponds to half of a counting period in Timer mode when the waveform is set to Square Wave and to the end of the Ton phase in PWM mode.

In order to be active, the PWM/Timers interrupts must be enabled by writing the Interrupt Mask Register (INT_MASK) in the Configuration Register Space, bits MSKT0 And MSKT1.

11.5 PWM/Timer 0 Register Description

The following registers are related to the use of the PWM/Timer 0.

11.5.1 PWM/Timer 0 Configuration Registers.

PWM/Timer 0 Control Register 1 (PWM0_CR1)

Configuration Register 9 (09h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
T0MOD	T0IES	T0IEF	T0IER	STRMOD	T0STRT	RESMOD	T0RES

Bit 7: **T0MOD** PWM/Timer 0 Mode
0: Timer Mode
1: PWM Mode

Bit 6: **T0IES** Interrupt on Stop signal Enable
0: interrupt disabled
1: interrupt enabled

Bit 5: **T0IEF** Interrupt on T0OUT falling Enable
0: interrupt disabled
1: interrupt enabled

Bit 4: **T0IES** Interrupt on T0OUT rising Enable
0: interrupt disabled
1: interrupt enabled

Bit 3: **STRMOD** Start signal mode
0: start on level
1: start on edge

Bit 2: **T0STRT** PWM/Timer 0 Start bit
0: Timer 0 stopped
1: Timer 0 started

Bit 1: **RESMOD** Reset signal mode
0: start on level
1: start on edge

Bit 0: **T0RES** PWM/Timer 0 Reset bit
0: PWM/Timer 0 reset
1: PWM/Timer 0 set

PWM/Timer 0 Control Register 2 (PWM0_CR2)
Configuration Register 10 (0Ah) Read/Write
Reset Value: 0000 0000 (00h)

7			4 … 0
-	-	T0WAV	T0PRESC

Bit 7-6: Not Used

Bit 5: **T0WAV** T0OUT Waveform
- 0: pulse (type2)
- 1: square (type1)

Bit 4-0: **T0PRESC** PWM/Timer 0 Prescaler
The PWM/Timer 0 clock is divided by a factor equal to $2^{T0PRESC}$. **The maximum value allowed for T0PRESC is 10000 (010h)**.

PWM/Timer 0 Control Register 3 (PWM0_CR3)
Configuration Register 11 (0Bh) Read/Write
Reset Value: 0000 0000 (00h)

7					0
T0SYNC	-	T1SYNC	T0CKS	STRSRC	RESSRC

Bit 7: **T0SYNC** PWM/Timer 0 Set/Reset mask
- 0: Set/Reset activated
- 1: Set/Reset masked

Bit 6: not used

Bit 5: **T1SYNC** PWM/Timer 1 Set/Reset mask
- 0: Set/Reset activated
- 1: Set/Reset masked

Bit 4: **T0CKS** PWM/Timer 0 Clock Source
- 0: Internal clock
- 1: External Clock from T0CLK

Bit 3-2: **STRSRC** PWM/Timer 0 Start signal source
- 00: Internal from T0STRT bit
- 01: External from T0STRT pin
- 10: Both internal and external

Bit 3-2: **RESSRC** PWM/Timer 0 Reset source
- 00: Internal from T0STRT bit
- 01: External from T0STRT pin
- 10: Both internal and external

Interrupt Polarity Register (INT_POL)
Configuration Register 1 (01h) Read/Write
Reset Value: 0000 0000 (00h)

7							0
-	-	LVD_EN	RESPOL	STRPOL	POLPB	POLPA	POLNMI

Bit 7-6: Not Used

Bit 5: See LVD Registers Description

Bit 4: **RESPOL** Reset signal polarity
- 0: Reset on low level/falling edge
- 1: Reset on high level/rising edge

Bit 3: **STRPOL** Start signal polarity
- 0: Start on high level/rising edge
- 1: Start on low level/falling edge

Bit 2-0: See Interrupt Registers Description

11.5.2 PWM/Timer 0 Input Registers.

PWM/Timer 0 Counter High Input Register (PWM0_COUNT_IN_H)
Input Register 21 (015h) Read only
Reset Value: 0000 0000 (00h)

7							0
T0CI15	T0CI14	T0CI13	T0CI12	T0CI11	T0CI10	T0CI9	T0CI8

Bit 7-0: **T0CI15-8** PWM/Timer 0 Counter MSB

In this register the current value of the Timer 0 Counter MSB can be read.

PWM/Timer 0 Counter Low Input Register (PWM0_COUNT_IN_L)

Input Register 22 (016h) Read only

Reset Value: 0000 0000 (00h)

7							0
T0CI7	T0CI6	T0CI5	T0CI4	T0CI3	T0CI2	T0CI1	T0CI0

Bit 7-0: **T0CI7-0** PWM/Timer 0 Counter MSB

In this register the current value of the Timer 0 Counter LSB can be read.

PWM/Timer 0 Status Register (PWM0_STATUS)

Input Register 23 (017h) Read only

Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	T0OVFL	T0OUT	T0RST	T0SST

Bit 7-4: Not Used

Bit 3: **T0OVFL** PWM**/**Timer 0 counter overflow flag
- 0: no overflow occurred since last reset
- 1: overflow occurred

Bit 2: **T0OUT** T0OUT pin value
- 0: T0OUT pin is at logical level 0
- 1: T0OUT pin is at logical level 1

Bit 2: **T0RST** Reset Status
- 0: PWM/Timer 0 is reset
- 1: PWM/Timer 0 is set

Bit 2: **T0SST** Start Status
- 0: PWM/Timer 0 is stopped
- 1: PWM/Timer 0 is running

PWM/Timer 0 Capture High Input Register (PWM0_CAPTURE_H)

Input Register 24 (018h) Read only

Reset Value: 0000 0000 (00h)

7							0
T0CP15	T0CP14	T0CP13	T0CP12	T0CP11	T0CP10	T0CP9	T0CP8

Bit 7-0: **T0CP15-8** PWM/Timer 0 Capture MSB

In this register the counter value after the last stop can be read.

PWM/Timer 0 Capture Low Input Register (PWM0_CAPTURE_L)

Input Register 25 (019h) Read only

Reset Value: 0000 0000 (00h)

7							0
T0CP7	T0CP6	T0CP5	T0CP4	T0CP3	T0CP2	T0CP1	T0CP0

Bit 7-0: **T0CP7-0** PWM/Timer 0 Capture LSB

In this register the counter value after the last stop can be read.

11.5.3 PWM/Timer 0 Output Registers.

PWM/Timer 0 Counter High Output Register (PWM0_COUNT_OUT_H)

Output Register 7 (07h) Write only

Reset Value: 0000 0000 (00h)

7							0
T0CO15	T0CO14	T0CO13	T0CO12	T0CO11	T0CO10	T0CO9	T0CO8

Bit 7-0: **T0CO15-8** PWM/Timer 0 Counter MSB

This register is used to write the Timer 0 Counter value (MSB).

Note: *this register is latched after writing the LSB part (PWM_COUNT_OUT_L: see below). For this reason this register must be written before the LSB.*

PWM/Timer 0 Counter Low Output Register (PWM0_COUNT_OUT_L)

Output Register 8 (08h) Write only

Reset Value: 0000 0000 (00h)

7							0
T0CO7	T0CO6	T0CO5	T0CO4	T0CO3	T0CO2	T0CO1	T0CO0

Bit 7-0: **T0CO7-0** PWM/Timer 0 Counter MSB

This register is used to write the Timer 0 Counter value (LSB).

Note: *writing this register, the PWM0_COUNT_OUT_x couple is latched in the internal registers of the peripherals. For this reason, this register should be written after the MSB one.*

PWM/Timer 0 Reload High Output Register (PWM0_RELOAD_H)

Output Register 9 (09h) Write only

Reset Value: 0000 0000 (00h)

7							0
T0REL15	T0REL14	T0REL13	T0REL12	T0REL11	T0REL10	T0REL9	T0REL8

Bit 7-0: **T0REL15-8** PWM/Timer 0 Reload MSB

This register is used to write the Timer 0 Reload value (MSB).

Note: *this register is latched after writing the LSB part (PWM0_RELOAD_L: see below). For this reason, this register must be written before the LSB.*

PWM/Timer 0 Reload Low Output Register (PWM0_RELOAD_L)

Output Register 8 (08h) Write only

Reset Value: 0000 0000 (00h)

7							0
T0REL7	T0REL6	T0REL5	T0REL4	T0REL3	T0REL2	T0REL1	T0REL0

Bit 7-0: **T0REL7-0** PWM/Timer 0 Reload LSB

This register is used to write the Timer 0 Reload value (LSB).

Note: *by writing this register, the PWM0_RELOAD_x couple is latched in the internal registers of the peripherals. For this reason this register should be written after the MSB one.*

11.6 PWM/Timer 1 Register Description

The following registers are related to the use of the PWM/Timer 1.

11.6.1 PWM/Timer 1 Configuration Registers.

PWM/Timer 1 Control Register 1 (PWM1_CR1)

Configuration Register 12 (0Ch) Read/Write

Reset Value: 0000 0000 (00h)

7							0
T0MOD	T0IES	T0IEF	T0IER	-	T0STRT	-	T0RES

Bit 7: **T1MOD** PWM/Timer 1 Mode
0: Timer Mode
1: PWM Mode

Bit 6: **T1IES** Interrupt on Stop signal Enable
0: interrupt disabled
1: interrupt enabled

Bit 5: **T1IEF** Interrupt on T1OUT falling Enable
0: interrupt disabled
1: interrupt enabled

Bit 4: **T1IES** Interrupt on T1OUT rising Enable
0: interrupt disabled
1: interrupt enabled

Bit 3: not used

Bit 2: **T1STRT** PWM/Timer 1 Start bit
0: Timer 0 stopped
1: Timer 0 started

Bit 1: not used

Bit 0: **T1RES** PWM/Timer 1 Reset bit
0: PWM/Timer 0 reset
1: PWM/Timer 0 set

PWM/Timer 1 Control Register 2 (PWM1_CR2)
Configuration Register 13 (0Dh) Read/Write
Reset Value: 0000 0000 (00h)

7			4				0
-	-	T1WAV	T1PRESC				

Bit 7-6: Not Used

Bit 5: **T1WAV** T1OUT Waveform
0: pulse (type2)
1: square (type1)

Bit 4-0: **T1PRESC** PWM/Timer 1 Prescaler
The PWM/Timer 1 clock is divided by a factor equal to $2^{T1PRESC}$. **The maximum value allowed for T1PRESC is 10000 (010h)**.

11.6.2 PWM/Timer 1 Input Registers.

PWM/Timer 1 Counter High Input Register (PWM1_COUNT_IN_H)
Input Register 26 (01Ah) Read only
Reset Value: 0000 0000 (00h)

7							0
T1CI15	T1CI14	T1CI13	T1CI12	T1CI11	T1CI10	T1CI9	T1CI8

Bit 7-0: **T1CI15-8** PWM/Timer 1 Counter MSB

In this register the current value of the Timer 1 Counter MSB can be read.

PWM/Timer 1 Counter Low Input Register (PWM1_COUNT_IN_L)
Input Register 27 (01Bh) Read only
Reset Value: 0000 0000 (00h)

7							0
T1CI7	T1CI6	T1CI5	T1CI4	T1CI3	T1CI2	T1CI1	T1CI0

Bit 7-0: **T1CI7-0** PWM/Timer 1 Counter MSB

In this register the current value of the Timer 0 Counter LSB can be read.

PWM/Timer 1 Status Register (PWM1_STATUS)
Input Register 28 (01Ch) Read only
Reset Value: 0000 0000 (00h)

7							0
-	-	-	-	T1OVFL	T1OUT	T1RST	T1SST

Bit 7-4: Not Used

Bit 3: **T1OVFL** PWM/Timer 0 counter overflow flag
0: no overflow occurred since last reset
1: overflow occurred

Bit 2: **T1OUT** T0OUT pin value
0: T0OUT pin is at logical level 0
1: T0OUT pin is at logical level 1

Bit 2: **T1RST** Reset Status
0: PWM/Timer 0 is reset
1: PWM/Timer 0 is set

Bit 2: **T1SST** Start Status
0: PWM/Timer 0 is stopped
1: PWM/Timer 0 is running

PWM/Timer 1 Capture High Input Register (PWM1_CAPTURE_H)
Input Register 29 (01Dh) Read only
Reset Value: 0000 0000 (00h)

7							0
T1CP15	T1CP14	T1CP13	T1CP12	T1CP11	T1CP10	T1CP9	T1CP8

Bit 7-0: **T1CP15-8** PWM/Timer 1 Capture MSB

In this register the counter value after the last stop can be read.

PWM/Timer 1 Capture Low Input Register (PWM1_CAPTURE_L)
Input Register 30 (01Eh) Read only

Reset Value: 0000 0000 (00h)

7							0
T1CP7	T1CP6	T1CP5	T1CP4	T1CP3	T1CP2	T1CP1	T1CP0

Bit 7-0: **T1CP7-0** PWM/Timer 1 Capture LSB

In this register the counter value after the last stop can be read.

11.6.3 PWM/Timer 1 Output Registers.

PWM/Timer 1 Counter High Output Register (PWM1_COUNT_OUT_H)

Output Register 11 (0Bh) Write only

Reset Value: 0000 0000 (00h)

7							0
T1CO15	T1CO14	T1CO13	T1CO12	T1CO11	T1CO10	T1CO9	T1CO8

Bit 7-0: **T1CO15-8** PWM/Timer 1 Counter MSB

This register is used to write the Timer 1 Counter value (MSB).

Note: *this register is latched after writing the LSB part (PWM1_COUNT_OUT_L: see below). For this reason, this register must be written before the LSB.*

PWM/Timer 1 Counter Low Output Register (PWM1_COUNT_OUT_L)

Output Register 12 (0Ch) Write only

Reset Value: 0000 0000 (00h)

7							0
T1CO7	T1CO6	T1CO5	T1CO4	T1CO3	T1CO2	T1CO1	T1CO0

Bit 7-0: **T1CO7-0** PWM/Timer 0 Counter MSB

This register is used to write the Timer 1 Counter value (LSB).

Note: *by writing this register, the PWM1_COUNT_OUT_x couple is latched in the internal registers of the peripherals. For this reason this register should be written after the MSB one.*

PWM/Timer 1 Reload High Output Register (PWM1_RELOAD_H)

Output Register 13 (0Dh) Write only

Reset Value: 0000 0000 (00h)

7							0
T1REL15	T1REL14	T1REL13	T1REL12	T1REL11	T1REL10	T1REL9	T1REL8

Bit 7-0: **T1REL15-8** PWM/Timer 0 Reload MSB

This register is used to write the Timer 1 Reload value (MSB).

Note: *this register is latched after writing the LSB part (PWM1_RELOAD_L: see below). For this reason, this register must be written before the LSB.*

PWM/Timer 1 Reload Low Output Register (PWM0_RELOAD_L)

Output Register 14 (0Eh) Write only

Reset Value: 0000 0000 (00h)

7							0
T1REL7	T1REL6	T1REL5	T1REL4	T1REL3	T1REL2	T1REL1	T01REL0

Bit 7-0: **T1REL7-0** PWM/Timer 1 Reload LSB

This register is used to write the Timer 1 Reload value (LSB).

Note: *by writing this register, the PWM1_RELOAD_x couple is latched in the internal registers of the peripherals. For this reason, this register should be written after the MSB one.*

12 I²C BUS INTERFACE (I²C)

12.1 Introduction

The I²C Bus Interface serves as an interface between the microcontroller and the serial I²C bus, providing both multimaster and slave functions and controls all I²C bus-specific sequencing, protocol, arbitration and timing. The I²Bus Interface supports fast I²C mode (400kHz).

12.2 Main Features

- Parallel-bus/I²C protocol converter
- Multi-master capability
- 7-bit/10-bit Addressing
- Transmitter/Receiver flag
- End-of-byte transmission flag
- Transfer problem detection

I²C Master Features:

- Clock generation
- I²C bus busy flag
- Arbitration Lost Flag
- End of byte transmission flag
- Transmitter/Receiver Flag
- Start bit detection flag
- Start and Stop generation

I²C Slave Features:

- Stop bit detection
- I²C bus busy flag
- Detection of misplaced start or stop condition
- Programmable I²C Address detection
- Transfer problem detection
- End-of-byte transmission flag
- Transmitter/Receiver flag

12.3 General Description

In addition to receiving and transmitting data, this interface converts it from serial to parallel format and vice versa, using either an interrupt or polled handshake. The interrupts are enabled or disabled via software. The interface is connected to the I²C bus by a data pin (SDA) and by a clock pin (SCL).

The interface can be connected both with a standard I²C bus and a Fast I²C bus. This selection is made via software.

12.3.1 Mode Selection.

The interface can operate in the following four modes:

– Slave transmitter/receiver

– Master transmitter/receiver

By default, it operates in slave mode.

The interface automatically switches from slave to master after it generates a START condition and from master to slave in case of arbitration loss or a STOP generation, providing Multi-Master capability.

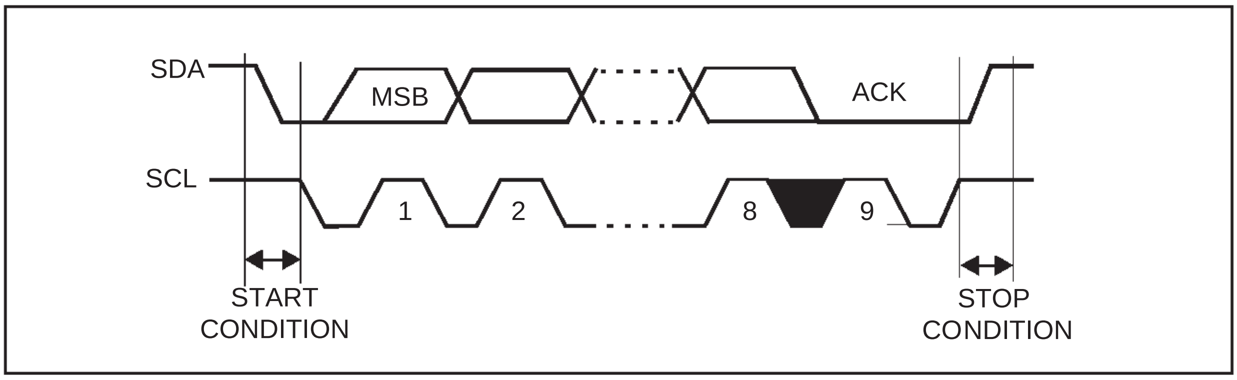
12.3.2 Communication Flow.

In Master mode, Communication Flow initiates data transfer and generates the clock signal. A serial data transfer always begins with a start condition and ends with a stop condition. Both start and stop conditions are generated in master mode by software.

In Slave mode the interface is capable of recognizing its own address (7 or 10-bit) and the General Call address. The General Call address detection may be enabled or disabled by software.

Data and addresses are transferred as 8-bit bytes, (MSB first). The first byte(s) follow the start condition is the address (one in 7-bit mode, two in 10-bit mode), which is always transmitted in Master mode.A 9th clock pulse follows the 8 clock cycles of a byte transfer, during which the receiver must send an acknowledge bit to the transmitter. Refer to Figure 12.1.

Figure 12.1 I²C BUS Protocol



Acknowledge may be enabled and disabled via software.

The I^{2}C interface address and/or general call address can be selected via software.

The speed of the I^{2}C interface may be selected between Standard (0-100KHz) and Fast I^{2}C (100-400KHz).

12.3.3 SDA/SCL Line Control.

Transmitter mode: the interface holds the clock line low before transmission, in order to wait for the microcontroller to write the byte in the Data Register.

Receiver mode: the interface holds the clock line low after reception to wait for the microcontroller to read the byte in the Data Register.

SCL frequency is controlled by a programmable clock divider which depends on the I^{2}C bus mode.

When the I^{2}C cell is enabled, the SDA and SCL pins must be configured as floating open-drain I/O.

The value of the external pull-up resistance used depends on the application.

12.4 Functional Description

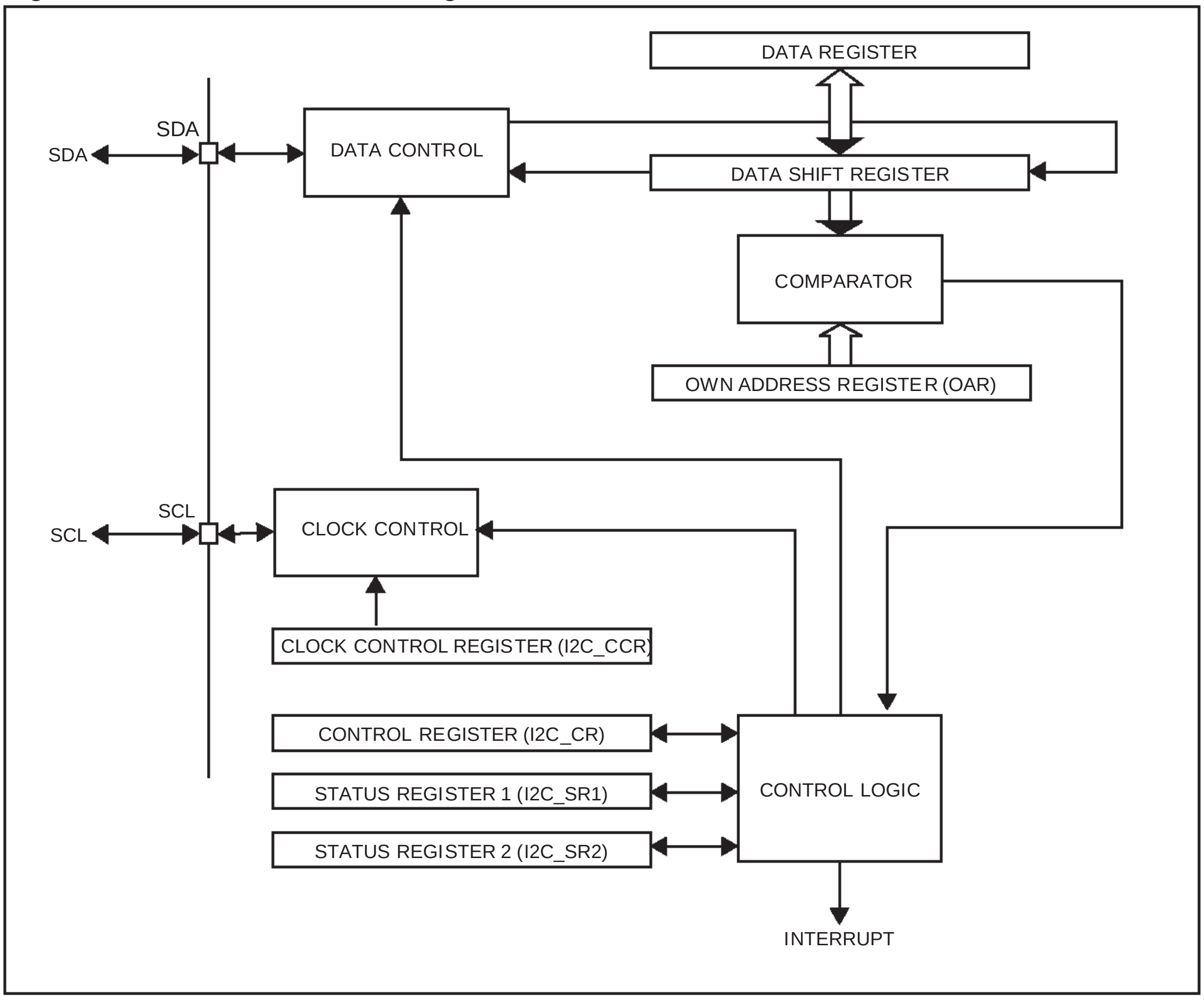
By default the I^{2}C interface operates in Slave mode (M/SL bit is cleared) except when it initiates a transmit or receive sequence.

First, the interface frequency must be configured using the related bits of the Configuration Registers.

12.4.1 Slave Mode.

As soon as a start condition is detected, the address is received from the SDA line and sent to the shift register; then it is compared with the address of the interface or the General Call address (if selected by software).

Figure 12.2 I^{2}C Interface Block Diagram


DATA REGISTER
SDA
SDA
DATA CONTROL
DATA SHIFT REGISTER
COMPARATOR
OWN ADDRESS REGISTER (OAR)
SCL
SCL
CLOCK CONTROL
CLOCK CONTROL REGISTER (I2C_CCR)
CONTROL REGISTER (I2C_CR)
STATUS REGISTER 1 (I2C_SR1)
STATUS REGISTER 2 (I2C_SR2)
CONTROL LOGIC
INTERRUPT

***Note:** In 10-bit addressing mode, the comparison includes the header sequence (11110xx0) and the two most significant bits of the address.*

Header matched (10-bit mode only): the interface generates an acknowledgement pulse if the ACK bit is set.

Address not matched: the interface ignores it and waits for another Start condition.

Address matched: the interface generates in sequence:

– Acknowledge pulse if the ACK bit is set.

– EVF and ADSL bits are set with an interrupt if the ITE bit is set.

Afterwards, the interface waits for the I2C_SR1 register to be read, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV1).

Next, in 7-bit mode read the I2C_IN register to determine from the least significant bit (Data Direction Bit) if the slave must enter Receiver or Transmitter mode.

In 10-bit mode, after receiving the address sequence the slave is always in receive mode. It will enter transmit mode on receiving a repeated Start condition followed by the header sequence with matching address bits and the least significant bit set (11110xx1).

Slave Receiver

Following reception of the address and after the I2C_SR1 register has been read, the slave receives bytes from the SDA line into the I2C_IN register via the internal shift register. After each byte, the interface generates the following in sequence:

– Acknowledge pulse if the ACK bit is set

– EVF and BTF bits are set with an interrupt if the ITE bit is set.

Afterwards, the interface waits for the I2C_SR1 register to be read followed by a read of the I2C_IN register, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV2).

Slave Transmitter

Following the address reception and after the I2C_SR1 register has been read, the slave sends bytes from the I2C_OUT register to the SDA line via the internal shift register.

The slave waits for a read of the I2C_SR1 register followed by a write in the I2C_OUT register, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV3).

When the acknowledge pulse is received:

– The EVF and BTF bits are set by hardware with an interrupt if the ITE bit is set.

Closing slave communication

After the last data byte is transferred a Stop Condition is generated by the master. The interface detects this condition and sets:

– EVF and STOPF bits with an interrupt if the ITE bit is set.

Afterwards, the interface waits for a read of the I2C_SR2 register (see Figure 12.3 Transfer sequencing EV4).

Error Cases

– **BERR**: Detection of a Stop or a Start condition during a byte transfer. In this case, the EVF and the BERR bits are set with an interrupt if the ITE bit is set.

If it is a Stop then the interface discards the data, released the lines and waits for another Start condition.

If it is a Start then the interface discards the data and waits for the next slave address on the bus.

– **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set with an interrupt if the ITE bit is set.

***Note:** In both cases, the SCL line is not held low; however, SDA line can remain low due to possible «0» bits transmitted last. At this point, both lines must be released by software.*

How to release the SDA / SCL lines

Set and subsequently clear the STOP bit while BTF is set. The SDA/SCL lines are released after the current byte is transferred.

12.4.2 Master Mode.

To switch from default Slave mode to Master mode a Start condition generation is needed.

Start condition

Setting the START bit while the BUSY bit is cleared causes the interface to switch to Master mode (M/SL bit set) and generates a Start condition.

Once the Start condition is sent:

– The EVF and SB bits are set by hardware with an interrupt if the ITE bit is set.

Afterwards, the master waits for a read of the I2C_SR1 register followed by a write in the I2C_OUT register with the Slave address, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV5).

Slave address transmission

At this point, the slave address is sent to the SDA line via the internal shift register.

In 7-bit addressing mode, one address byte is sent.

In 10-bit addressing mode, sending the first byte including the header sequence causes the following event:

– The EVF bit is set by hardware with interrupt generation if the ITE bit is set.

Afterwards, the master waits for a read of the I2C_SR1 register followed by a write in the I2C_OUT register, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV9).

The second address byte is sent by the interface.

After completion of this transfer (and acknowledge from the slave if the ACK bit is set):

– The EVF bit is set by hardware with interrupt generation if the ITE bit is set.

Afterwards, the master waits for a read of the I2C_SR1 register followed by a write in the I2C_CR register (for example set PE bit), **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV6).

Next, the master must enter Receiver or Transmitter mode.

Note: *In 10-bit addressing mode, in order to switch the master to Receiver mode, software must generate a repeated Start condition and resend the header sequence with the least significant bit set (11110xx1).*

Master Receiver

Following the address transmission and after I2C_SR1 and I2C_CR registers have been accessed, the master receives bytes from the SDA line into the I2C_IN register via the internal shift register. After each byte the interface generates in sequence:

– Acknowledge pulse if the ACK bit is set

– EVFand BTF bits are set by hardware with an interrupt if the ITE bit is set.

Afterwards, the interface waits for a read of the I2C_SR1 register followed by a read of the I2C_IN register, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV7).

In order to close the communication: before reading the last byte from the I2C_IN register, set the STOP bit to generate the Stop condition. The interface automatically goes back to slave mode (M/SL bit cleared).

Note: *In order to generate the non-acknowledge pulse after the last data byte received, the ACK bit must be cleared just before reading the second last data byte.*

Master Transmitter

Following the address transmission and after the I2C_SR1 register has been read, the master sends bytes from the I2C_OUT register to the SDA line via the internal shift register.

The master waits for a read of the I2C_SR1 register followed by a write in the I2C_OUT register, **holding the SCL line low** (see Figure 12.3 Transfer sequencing EV8).

When the acknowledge bit is received, the interface sets:

– EVF and BTF bits with an interrupt if the ITE bit is set.

In order to close the communication: after writing the last byte to the I2C_OUT register, set the STOP bit to generate the Stop condition. The interface automatically returns to slave mode (M/SL bit cleared).

Error Cases

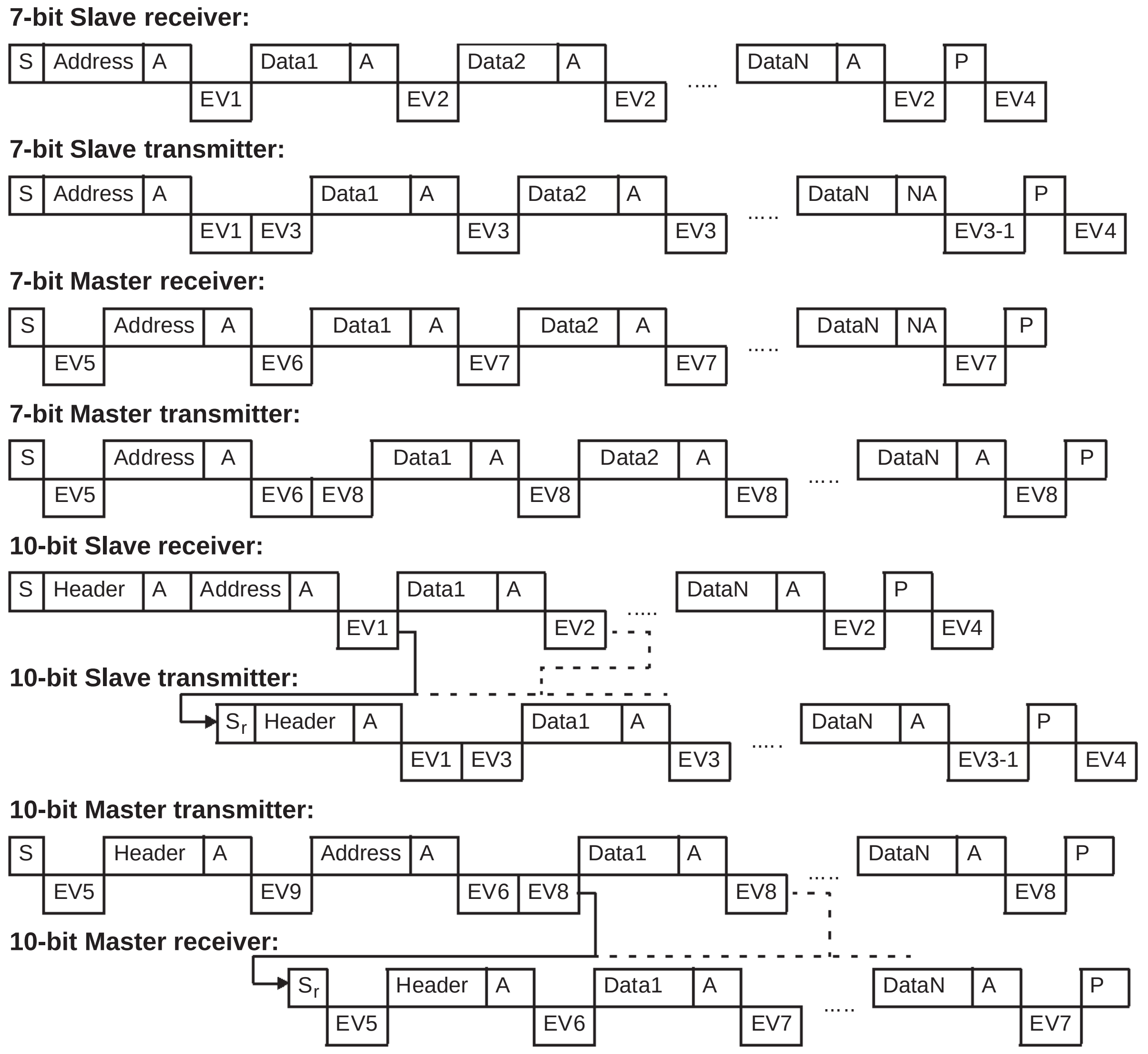
– **BERR**: Detection of a Stop or a Start condition during a byte transfer. In this case, the EVF and BERR bits are set by hardware with an interrupt if ITE is set.

– **AF**: Detection of a non-acknowledge bit. In this case, the EVF and AF bits are set by hardware with an interrupt if the ITE bit is set. To resume, set the START or STOP bit.

– **ARLO**: Detection of an arbitration lost condition. In this case the ARLO bit is set by hardware (with an interrupt if the ITE bit is set and the interface automatically goes back to slave mode (the M/SL bit is cleared).

Note: *In all these cases, the SCL line is not held low; however, the SDA line can remain low due to possible «0» bits transmitted last. Both lines must be released via software.*

Figure 12.3 Tranfer Sequencing


7-bit Slave receiver:
S Address A EV1 Data1 A EV2 Data2 A EV2 ..... DataN A EV2 P EV4
7-bit Slave transmitter:
S Address A EV1 EV3 Data1 A EV3 Data2 A EV3 ..... DataN NA EV3-1 P EV4
7-bit Master receiver:
S EV5 Address A EV6 Data1 A EV7 Data2 A EV7 ..... DataN NA EV7 P
7-bit Master transmitter:
S EV5 Address A EV6 EV8 Data1 A EV8 Data2 A EV8 ..... DataN A EV8 P
10-bit Slave receiver:
S Header A Address A EV1 Data1 A EV2 ..... DataN A EV2 P EV4
10-bit Slave transmitter:
Sr Header A EV1 EV3 Data1 A EV3 ..... DataN A EV3-1 P EV4
10-bit Master transmitter:
S EV5 Header A EV9 Address A EV6 EV8 Data1 A EV8 ..... DataN A EV8 P
10-bit Master receiver:
Sr EV5 Header A EV6 Data1 A EV7 ..... DataN A EV7 P

Legend:

S=Start, P=Stop, A=Acknowledge, NA=Non-acknowledge

EVx=Event (with interrupt if ITE=1)

EV1: EVF=1, ADSL=1, cleared by reading I2C_SR1 register.

EV2: EVF=1, BTF=1, cleared by reading I2C_SR1 register followed by reading I2C_IN register.

EV3: EVF=1, BTF=1, cleared by reading I2C_SR1 register followed by writing I2C_OUT register.

EV3-1: EVF=1, AF=1, BTF=1; AF is cleared by reading I2C_SR1. BTF is cleared by releasing the lines (STOP=1, STOP=0) or by writing I2C_OUT register (I2C_OUT=FFh). **Note:** If lines are released by STOP=1, STOP=0, the subsequent EV4 is not seen.

EV4: EVF=1, STOPF=1, cleared by reading I2C_SR2 register.

EV5: EVF=1, SB=1, cleared by reading I2C_SR1 register followed by writing I2C_OUT register.

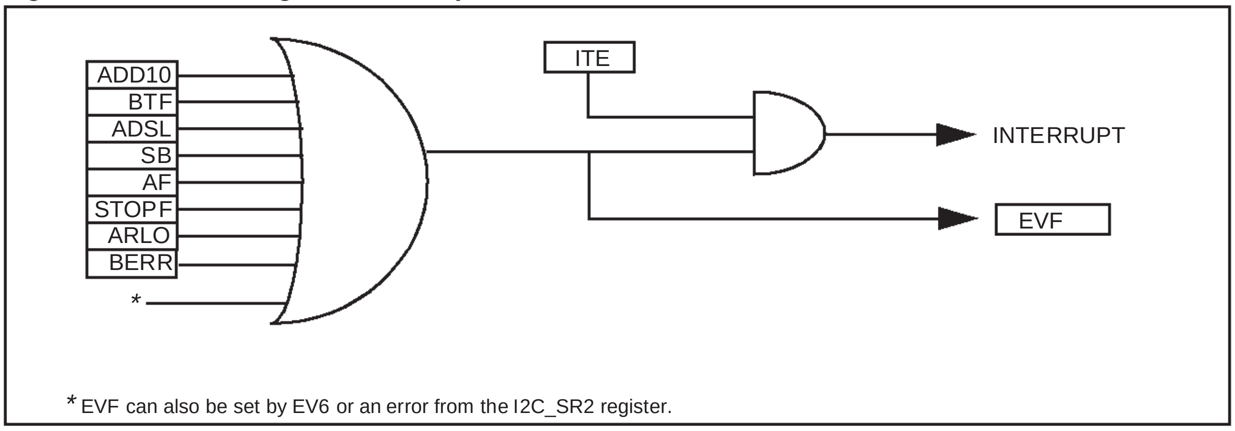
EV6: EVF=1, cleared by reading I2C_SR1 register followed by writing I2C_CR (for example PE=1

EV7: EVF=1, BTF=1, cleared by reading I2C_SR1 register followed by reading I2C_IIN register.

EV8: EVF=1, BTF=1, cleared by reading I2C_SR1 register followed by writing I2C_OUT register.

EV9: EVF=1, ADD10=1, cleared by reading I2C_SR1 register followed by writing I2C_OUT registe

Figure 12.4 Event Flags and Interrupt Generation


ADD10
BTF
ADSL
SB
AF
STOPF
ARLO
BERR
*
ITE
INTERRUPT
EVF
* EVF can also be set by EV6 or an error from the I2C_SR2 register.

Interrupt Event	Event Flag	Enable Control Bit	Exit from Wait	Exit from Halt
10-bit Address Sent Event (Master Mode)	ADD10	ITE	Yes	No
End of Byte Transfer Event	BTF		Yes	No
Address Matched Event (Slave Mode)	ADSEL		Yes	No
Start Bit Generation Event (Master Mode)	SB		Yes	No
Acknowledge Failure Event	AF		Yes	No
Stop Detection Event (Slave Mode)	STOPF		Yes	No
Arbitration Lost Event (Multimaster configuration)	ARLO		Yes	No
Bus Error Event	BERR		Yes	No

Note: *The I^2C interrupt events are connected to the same interrupt vector. They generate an interrupt if the corresponding Enable Control Bit (ITE) is set and the Interrupt Mask bit (MSKI2C) in the INT_MASK Configuration Register is unmasked (set to 1, see Interrupts Chapter).*

12.5 Register Description

In the following sections describe the registers used by the I^{2}C Interface are described.

12.5.1 I^{2}C Interface Configuration Registers.

I^{2}C Control Register (I2C_CR)

Configuration Register 16 (010h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
-	-	PE	ENGC	START	ACK	STOP	ITE

Bit 7-6: Not Used

Bit 5: **PE** Peripheral Enable.

This bit is set and cleared by software

0: peripheral disabled

1: peripheral enabled

Notes:

– When PE=0, all the bits of the I2C_CR register and the SR register except the Stop bit are reset. All outputs are released while PE=0

– When PE=1, the corresponding I/O pins are selected by hardware as alternate functions.

– To enable the I^{2}C interface, write the I2C_CR register **TWICE** with PE=1 as the first write only activates the interface (only PE is set).

Bit 4: **ENGC** Enable General Call

This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0).

0: General Call disabled

1: General Call enabled

Note: *The 00h General Call address is acknowledged (01h ignored).*

Bit 3: **START** Generation of a Start Condition

This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0) or when the Start condition is sent (with interrupt generation if ITE=1).

– In Master Mode

0: No Start generation

1: Repeated Start generation

– In Slave Mode

0: No Start generation

1: Start generation when the bus is free

Bit 2: **ACK** Acknowledge enable

This bit is set and cleared by software. It is also cleared by hardware when the interface is disabled (PE=0).

0: No acknowledge returned

1: Acknowledge returned after an address byte or a data byte is received

Bit 1: **STOP** Reset signal mode

This bit is set and cleared by software. It is also cleared by hardware in master mode. Note: This bit is not cleared when the interface is disabled (PE=0).

– In Master Mode

0: No Stop generation

1: Stop generation after the current byte transfer or after the current Start condition is sent. The STOP bit is cleared by hardware when the Stop condition is sent.

– In Slave Mode

0: No Start generation

1: Release the SCL and SDA lines after the current byte transfer (BTF=1). In this mode the STOP bit has to be cleared by software.

Bit 0: **ITE** Interrupt Enable

0: Interrupt disabled

1: Interrupt enabled

I^{2}C Clock Control Register (I2C_CCR)

Configuration Register 17 (011h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
FM/SM	CC6	CC5	CC4	CC3	CC2	CC1	CC0

Bit 7: **FM/SM** Fast/Standard I^{2}C Mode.

This bit is set and cleared by software. It is not cleared when the interface is disabled (PE=0).

0: Standard I^{2}C Mode

1: Fast I^{2}C Mode

Bit 6-0: **CC6-CC0** 7-bit clock divider

These bits select the speed of the bus (F_{SCL}) depending on the I^{2}C mode. They are not cleared when the interface is disabled (PE=0).

– Standard mode (FM/SM=0): F_{SCL} <= 100kHz
  $F_{SCL} = f_{CPU}/(3x([CC6..CC0]+9))$

– Fast mode (FM/SM=1): F_{SCL} > 100kHz
  $F_{SCL} = f_{CPU}/(2x([CC6..CC0]+7))$

Warning: *For safety reason, CC6-CC0 bits must be configured with a value >= 3 for the Standard mode and >=2 for the Fast mode.*

I^{2}C Own Address Register 1 (I2C_OAR1)

Configuration Register 18 (012h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
ADD7	ADD6	ADD5	ADD4	ADD3	ADD2	ADD1	ADD0

7-bit Addressing Mode

bit 7-1: **ADD7-ADD1** Interface address.

These bits define the I^{2}C bus address of the interface. They are not cleared when the interface is disabled (PE=0).

Bit 0: **ADD0** *Address direction bit.*

This bit is “don’t care”, the interface acknowledges either 0 or 1. It is not cleared when the interface is disabled (PE=0).

Note: *Address 01h is always ignored.*

10-bit Addressing Mode

bit 7-0: **ADD7-ADD0** Interface address.

These are the least significant bits of the I^{2}C bus address of the interface. They are not cleared when the interface is disabled (PE=0).

I^{2}C Own Address Register 2 (I2C_OAR2)

Configuration Register 19 (013h) Read/Write

Reset Value: 0000 0000 (00h)

7					2		0
-	-	-	-	-	ADD9	ADD8	-

Bit 7-3: Not Used

bit 7-1: **ADD8-ADD8** Interface address.

These are the most significant bits of th I^{2}C bus address of the interface (10-bit mode only). They are not cleared when the interface is disabled (PE=0).

Bit 0: Reserved

12.5.2 I^{2}C Interface Input Registers.

I^{2}C Data Input Register (I2C_IN)

Input Register 6 (06h) Read only

Reset Value: 0000 0000 (00h)

7							0
I2CDI7	I2CDI6	I2CDI5	I2CDI4	I2CDI3	I2CDI2	I2CDI1	I2CDI0

bit 7-0: **I2CDI7-I2CDI0** Received data.

These bits contain the byte to be received from the bus in Receiver mode: the first data byte is received automatically in the I2C_IN register using the least significant bit of the address.

Then, the next data bytes are received one by one after reading the I2C_IN register.

I^{2}C Status Register 1 (I2C_SR1)

Input Register 7 (07h) Read only

Reset Value: 0000 0000 (00h)

7							0
EVF	ADD10	TRA	BUSY	BTF	ADSL	M/SL	SB

Bit 7: **EVF** Event Flag

This bit is set by hardware as soon as an event occurs. It is cleared by software reading I2C_SR2 register in case of error event or as described in Figure 12.3. It is also cleared by hardware when the interface is disabled (PE=0).

0: No event

1: One of the following events has occurred:

– BTF=1 (Byte received or transmitted)
– ADSL=1 (Address matched in Slave mode while ACK=1)
– SB=1 (Start condition generated in Master mode)
– AF=1 (No acknowledge received after byte transmission)
– STOPF=1 (Stop condition detected in Slave mode)
– ARLO=1 (Arbitration lost in Master mode)
– BERR=1 (Bus error, misplaced Start or Stop condition detected)
– Address byte successfully transmitted in Master mode.

Bit 6: **ADD10** 10 bit addressing in Master Mode

This bit is set by hardware when the master has sent the first byte in 10-bit address mode. It is cleared by software reading I2C_SR2 register followed by a write in the I2C_OUT register of the second address byte. It is also cleared by hardware when the peripheral is disabled (PE=0).

0: No ADD10 event occurred

1: The Master has sent the first address byte

Bit 5: **TRA** Transmitter/Receiver

When BTF is set, TRA=1 if a data byte has been transmitted. It is cleared automatically when BTF is cleared. It is also cleared by hardware after detection of Stop condition (STOPF=1), loss of bus arbitration (ARLO=1) or when the interface is disabled (PE=0).

0: Data byte received (if BTF=1)

1: Data byte transmitted

Bit 4: **BUSY** Bus busy

This bit is set by hardware on detection of a Start condition and cleared by hardware on detection of a Stop condition. It indicates a communication in progress on the bus. This information is still updated when the interface is disabled (PE=0).

0: No communication on the bus

1: Communication ongoing on the bus

Bit 3: **BTF** Byte transfer finished

This bit is set by hardware as soon as a byte is correctly received or transmitted with interrupt generation if ITE=1. It is cleared by software reading I2C_SR1 register followed by a read of I2C_IN or write of I2C_OUT registers. It is also cleared by hardware when the interface is disabled (PE=0).

– Following a byte transmission, this bit is set after reception of the acknowledge clock pulse. In case an address byte is sent, this bit is set only after the EV6 event (see Figure 12.3). BTF is cleared by reading I2C_SR1 register followed by writing the next byte in I2C_OUT register.
– Following a byte reception, this bit is set after transmission of the acknowledge clock pulse if ACK=1. BTF is cleared by reading I2C_SR1 register followed by reading the byte from I2C_IN register.

The SCL line is held low while BTF=1.

0: Byte transfer not done

1: Byte transfer succeeded

Bit 2: **ADSL** Address matched (Slave Mode)

This bit is set by hardware as soon as the slave address received matched with the OAR register content or a general call is recognized. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR1 register or by hardware when the interface is disabled (PE=0).

The SCL line is held low while ADSL=1.

0: Address mismatched or not received

1: Received address matched

Bit 1: **M/SL** Master/Slave

This bit is set by hardware as soon as the interface is in Master mode (writing START=1). It is cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1). It is also cleared when the interface is disabled (PE=0).

0: Slave mode

1: Master mode

Bit 0: **SB** Start bit (Master Mode)

This bit is set by hardware as soon as the Start condition is generated (following a write

START=1). An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR1 register followed by writing the address byte in I2C_OUT register. It is also cleared by hardware when the interface is disabled (PE=0).

0: No Start condition

1: Start condition generated

I^{2}C Status Register 2 (I2C_SR2)

Input Register 8 (08h) Read only

Reset Value: 0000 0000 (00h)

7							0
-	-	-	AF	STOPF	ARLO	BERR	GCAL

Bit 7-5: Reserved.

Bit 4: **AF** *Acknowledge failure*.

This bit is set by hardware when an acknowledge is returned. An interrupt is generated if ITE=1. It is cleared by software reading the I2C_SR2 register or by hardware when the interface is disabled (PE=0).

The SCL line is not held low while AF=1.

0: No acknowledge failure

1: Acknowledge failure

Bit 3: **STOPF** *Stop detection (Slave mode).*

This bit is set by hardware when a Stop condition is detected on the bus after an acknowledge (if ACK=1). An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0).

The SCL line is not held low while STOPF=1.

0: No Stop condition detected

1: Stop condition detected

Bit 2: **ARLO** *Arbitration lost*.

This bit is set by hardware when the interface loses the arbitration of the bus to another master. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0).

After an ARLO event the interface switches back automatically to Slave mode (M/SL=0).

The SCL line is not held low while ARLO=1.

0: No arbitration lost detected

1: Arbitration lost detected

Bit 1: **BERR** *Bus error.*

This bit is set by hardware when the interface detects a misplaced Start or Stop condition. An interrupt is generated if ITE=1. It is cleared by software reading I2C_SR2 register or by hardware when the interface is disabled (PE=0).

The SCL line is not held low while BERR=1.

0: No misplaced Start or Stop condition

1: Misplaced Start or Stop condition

Bit 0: **GCAL** *General Call (Slave mode).*

This bit is set by hardware when a general call address is detected on the bus while ENGC=1. It is cleared by hardware detecting a Stop condition (STOPF=1) or when the interface is disabled (PE=0).

0: No general call address detected on bus

1: general call address detected on bus

12.5.3 I^{2}C Interface Output Registers.

I^{2}C Data Output Register (I2C_OUT)

Output Register 6 (06h) Read only

Reset Value: 0000 0000 (00h)

7							0
I2CDO7	I2CDO6	I2CDO5	I2CDO4	I2CDO3	I2CDO2	I2CDO1	I2CDO0

bit 7-0: **I2CDO7-I2CDO0** Data to be transmitted.

These bits contain the byte to be transmitted in the bus in Transmitter mode: Byte transmission start automatically when the software writes in the I2C_OUT register.

13 SERIAL PERIPHERAL INTERFACE (SPI)

13.1 Introduction

The Serial Peripheral Interface (SPI) allows full-duplex, synchronous, serial communication with external devices. An SPI system may consist of a master, one or more slaves, or a system, in which devices may be either masters or slaves.

SPI is normally used for communication between the ICU and external peripherals or another ICU.

Refer to the Pin Description section in this datasheet for the device-specific pin-out.

13.2 Main Features

- Full duplex, three-wire synchronous transfers
- Master or slave operation
- Four master mode frequencies
- Maximum slave mode frequency = CKM/4.
- Four programmable master bit rates
- Programmable clock polarity and phase
- End of transfer interrupt flag
- Write collision flag protection
- Master mode fault protection capability.

13.3 General description

SPI is connected to external devices through 4 alternate pins:

- MISO: Master In / Slave Out pin
- MOSI: Master Out / Slave In pin
- SCK: Serial Clock pin
- $\overline{\text{SS}}$: Slave select pin (if not done through software)

A basic example of interconnections between a single master and a single slave is illustrated in Figure 13.1

The MOSI pins are connected together as the MISO pins. In this manner, data is transferred serially between master and slave (most significant bit first).

When the master device transmits data to a slave device via the MOSI pin, the slave device responds by sending data to the master device via the MISO pin. This implies full duplex transmission with both data out and data in synchronized with the same clock signal (which is provided by the master device via the SCK pin).

The transmitted byte is replaced by the byte received and eliminates the need for separate transmit-empty and receiver-full bits. A status flag is used to indicate that the I/O operation is complete.

Four possible data/clock timing relationships may be chosen (see Figure 13.4), but master and slave must be programmed with the same timing mode.

13.4 Functional Description

Figure 13.2 shows the serial peripheral interface (SPI) block diagram.

This interface contains 3 dedicated registers:

- A Control Register (SPI_CR)
- A Status Register (SPI_STATUS_CR)
- A Data Register for transmission (SPI_OUT)
- A Data Register for reception (SPI_OUT)

13.4.1 Master Configuration.

In a master configuration, the serial clock is generated on the SCK pin.

Figure 13.1 SPI Master Slave

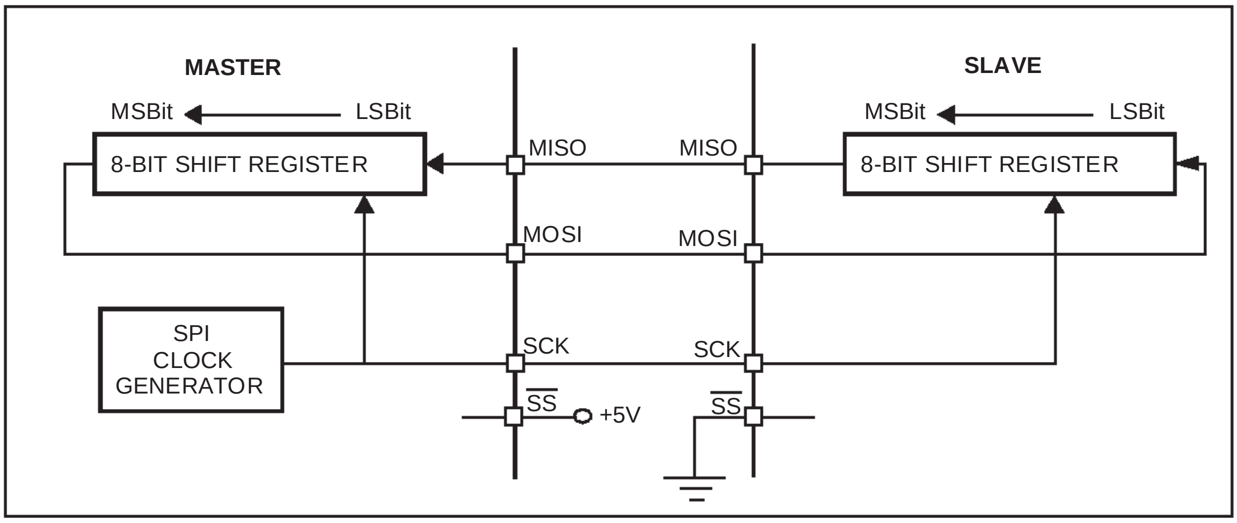
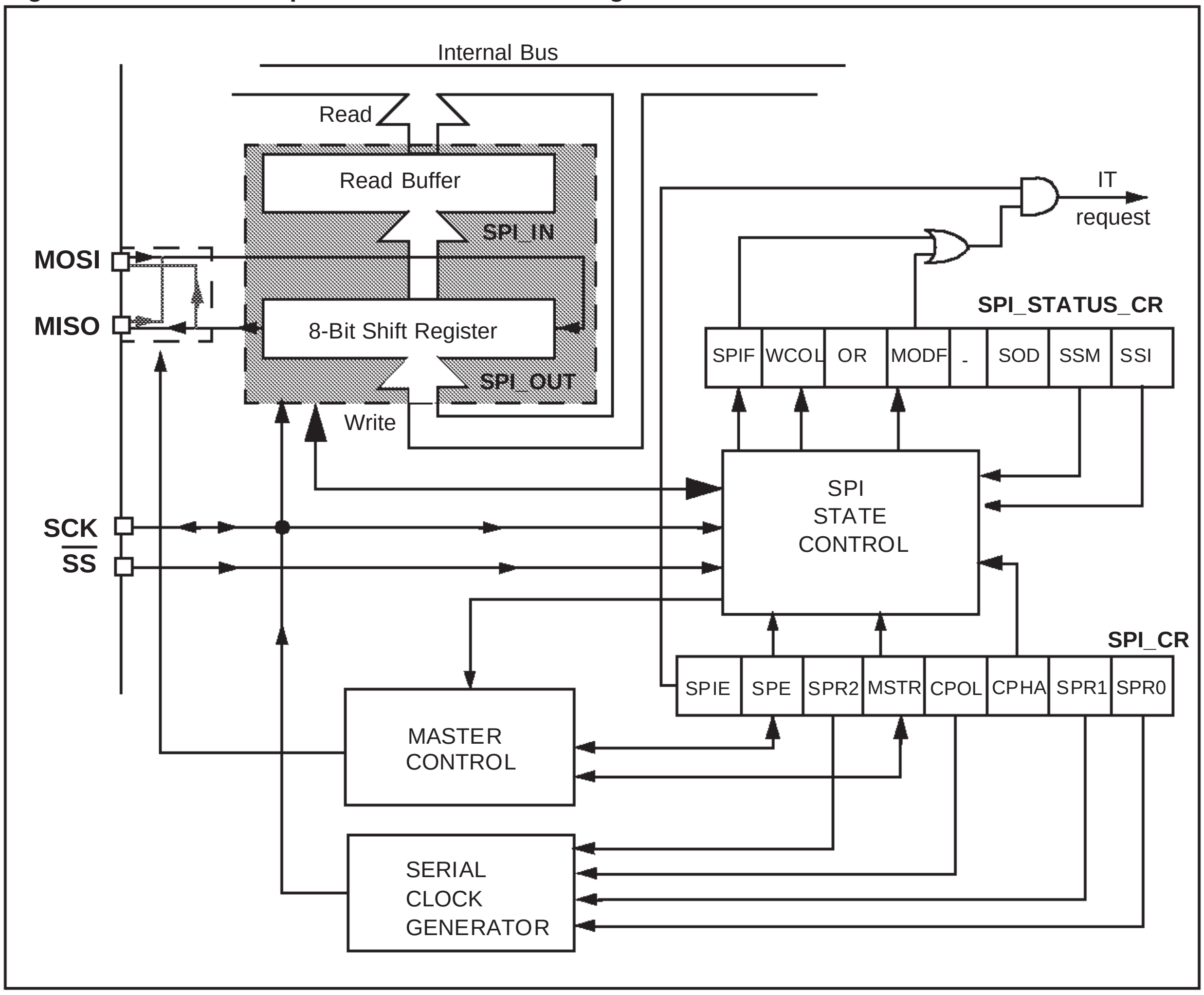


Figure 13.2 Serial Peripheral Interface Block Diagram



Procedure

– Select the SPR0, SPR1 and SPR2 bits to define the serial clock baud rate (see SPI_CR register).

– Select the CPOL and CPHA bits to define one of the four relationships between the data transfer and the serial clock (see Figure 13.4).

– The $\overline{SS}$ pin must be connected to a high level signal during the complete byte transmit sequence.

– The MSTR and SPE bits must be set (they remain set only if the $\overline{SS}$ pin is connected to a high level signal).

In this configuration the MOSI pin is a data output and to the MISO pin is a data input.

Transmit sequence

Transmit sequence begins when a byte is written in the SPI_OUT register.

The data byte is loaded in parallel into the 8-bit shift register (from the internal bus) during a write cycle and then shifted out serially to the MOSI pin most significant bit first.

When data transfer is complete:

– The SPIF bit is set by hardware

– An interrupt is generated if the SPIE bit is set.

During the last clock cycle the SPIF bit is set, a copy of the data byte received in the shift register is moved to a buffer. When the SPI_IN register is read, the SPI peripheral returns this buffered value. Clearing the SPIF bit is performed by the following software sequence:

1. An access to the SPI_STATUS_CR register while the SPIF bit is set
2. A read to the SPI_IN register.

Note: *While the SPIF bit is set, all writes to the SPI_OUT register are inhibited until the SPI_STATUS_CR register is read.*

13.4.2 Slave Configuration.

In slave configuration, the serial clock is received on the SCK pin from the master device.

The value of the SPR0, SPR1 and SPR2 bits is not used for data transfer.

Procedure

– For correct data transfer, the slave device must be in the same timing mode as the master device (CPOL and CPHA bits). See Figure 13.4.

– The $\overline{SS}$ pin must be connected to a low level signal during the complete byte transmit sequence.

– Clear the MSTR bit and set the SPE bit to assign the pins to alternate function.

In this configuration the MOSI pin is a data input and the MISO pin is a data output.

Transmit Sequence

The data byte is loaded into the 8-bit shift register (from the internal bus) during a write cycle and then shifted out serially to the MISO pin most significant bit first.

The transmit sequence begins when the slave device receives the clock signal and the most significant bit of the data on its MOSI pin.

When data transfer is complete:

– The SPIF bit is set by hardware

– An interrupt is generated if SPIE bit is set.

During the last clock cycle the SPIF bit is set, a copy of the data byte received in the shift register is moved to a buffer. When the SPI_IN register is read, the SPI peripheral returns the buffer value.

The SPIF bit is cleared by the following software sequence:

1. An access to the SPI_STATUS_CR register while the SPIF bit is set.
2. A read to the SPI_IN register.

***Note:** While the SPIF bit is set, all writes to the SPI_OUT register are inhibited until the SPI_STATUS_CR register is read.*

The SPIF bit can be cleared during a second transmission; however, it must be cleared before the second SPIF bit in order to prevent an overrun condition (see Section 13.4.6).

Depending on the CPHA bit, the $\overline{SS}$ pin has to be set to write to the SPI_OUT register between each data byte transfer to avoid a write collision (see Section 13.4.4).

13.4.3 Data Transfer Format.

During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). The serial clock is used to synchronize data transfer during a sequence of eight clock pulses.

The $\overline{SS}$ pin allows individual selection of a slave device; the other slave devices that are not selected do not interfere with SPI transfer.

Clock Phase and Clock Polarity

Four possible timing relationships may be chosen by software, using the CPOL and CPHA bits.

The CPOL (clock polarity) bit controls the steady state value of the clock when data isn’t being transferred. This bit affects both master and slave modes.

The combination between the CPOL and CPHA (clock phase) bits select the data capture clock edge.

Figure 13.4, shows an SPI transfer with the four combinations of the CPHA and CPOL bits. The diagram may be interpreted as a master or slave timing diagram where the SCK pin, the MISO pin, the MOSI pin are directly connected between the master and the slave device.

The $\overline{SS}$ pin is the slave device select input and can be driven by the master device.

The master device applies data to its MOSI pin-clock edge before the capture clock edge.

CPHA bit is set

The second edge on the SCK pin (falling edge if the CPOL bit is reset, rising edge if the CPOL bit is set) is the MSBit capture strobe. Data is latched on the occurrence of the second clock transition.

A write collision should not occur even if the $\overline{SS}$ pin stays low during a transfer of several bytes (see Figure 13.3).

CPHA bit is reset

The first edge on the SCK pin (falling edge if CPOL bit is set, rising edge if CPOL bit is reset) is the MSBit capture strobe. Data is latched on the occurrence of the first clock transition.

The $\overline{SS}$ pin must be toggled high and low between each byte transmitted (see Figure 13.3).

In order to protect the transmission from a write collision a low value on the $\overline{SS}$ pin of a slave device freezes the data in its SPI_OUT register and does not allow it to be altered. Therefore, the $\overline{SS}$ pin must be high to write a new data byte in the SPI_OUT without producing a write collision.

13.4.4 Write Collision Error.

A write collision occurs when the software tries to write to the SPI_OUT register while a data transfer is taking place with an external device. When this occurs, the transfer continues uninterrupted; and the software writing will be unsuccessful.

Write collisions can occur both in master and slave mode.

Note: *a “read collision” will never occur since the data byte received is placed in a buffer, in which access is always synchronous with the ICU operation.*

In Slave mode

When the CPHA bit is set:

The slave device will receive a clock (SCK) edge prior to the latch of the first data transfer. This first clock edge will freeze the data in the slave device SPI_OUT register and output the MSBit on to the external MISO pin of the slave device.

The $\overline{\text{SS}}$ pin low state enables the slave device, but the output of the MSBit onto the MISO pin does not take place until the first data transfer clock edge occurs.

When the CPHA bit is reset:

Data is latched on the occurrence of the first clock transition. The slave device doesn’t have a way of knowing when that transition will occur; therefore, the slave device collision occurs when software attempts to write the SPI_OUT register after its $\overline{\text{SS}}$ pin has been pulled low.

For this reason, the $\overline{\text{SS}}$ pin must be high, between each data byte transfer, in order to allow the CPU to write in the SPI_OUT register without generating a write collision.

In Master mode

Collision in the master device is defined as a write of the SPI_OUT register, while the internal serial clock (SCK) is in the process of transfer.

The $\overline{\text{SS}}$ pin signal must always be high on the master device.

WCOL bit

The WCOL bit in the SPI_STATUS_CR register is set if a write collision occurs.

No SPI interrupt is generated when the WCOL bit is set (the WCOL bit is a status flag only).

The WCOL bit is cleared by a software sequence (see Section 13.5).

13.4.5 Master Mode Fault.

Master mode fault occurs when the master device has its $\overline{\text{SS}}$ pin pulled low, then the MODF bit is set.

Master mode fault affects the SPI peripheral in the following ways:

– The MODF bit is set and an SPI interrupt is generated if the SPIE bit is set.

– The SPE bit is reset. This blocks all output from the device and disables the SPI peripheral.

– The MSTR bit is reset, forcing the device into slave mode.

Clearing the MODF bit is done through a software sequence:

1. A read or write access to the SPI_STATUS_CR register while the MODF bit is set.
2. A write to the SPI_CR register.

Note: *To avoid any multiple slave conflicts in the case of a system comprising several MCUs, the $\overline{\text{SS}}$ pin must be pulled high during the clearing sequence of the MODF bit. The SPE and MSTR bits may be restored to their original state during or after this clearing sequence.*

Hardware does not allow the user to set the SPE and MSTR bits, while the MODF bit is set (except in the MODF bit clearing sequence).

In a slave device the MODF bit can’t be set, but in a multi master configuration the device can be in slave mode with this MODF bit set.

The MODF bit indicates that there might have been a multi-master conflict for system control and allows a proper exit from system operation to a reset or default system state using an interrupt routine.

Figure 13.3 CHPA/$\overline{\text{SS}}$ Timing Diagram

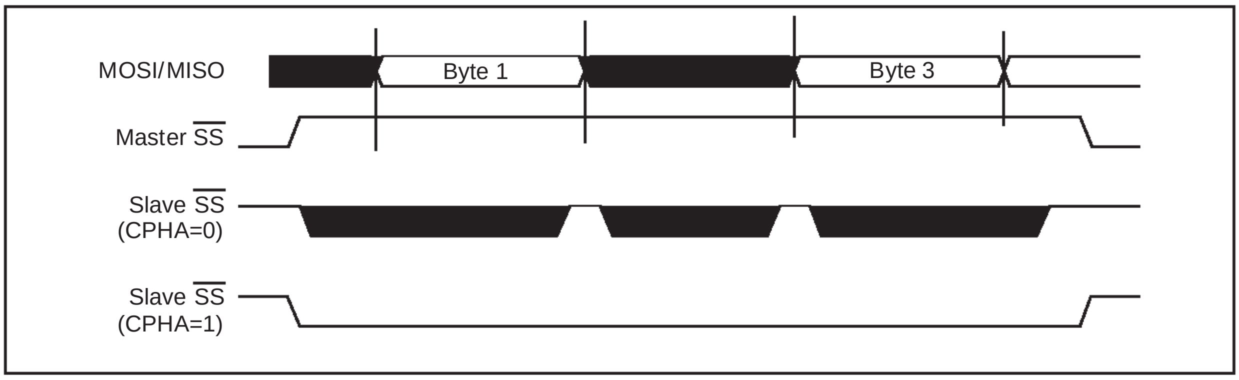
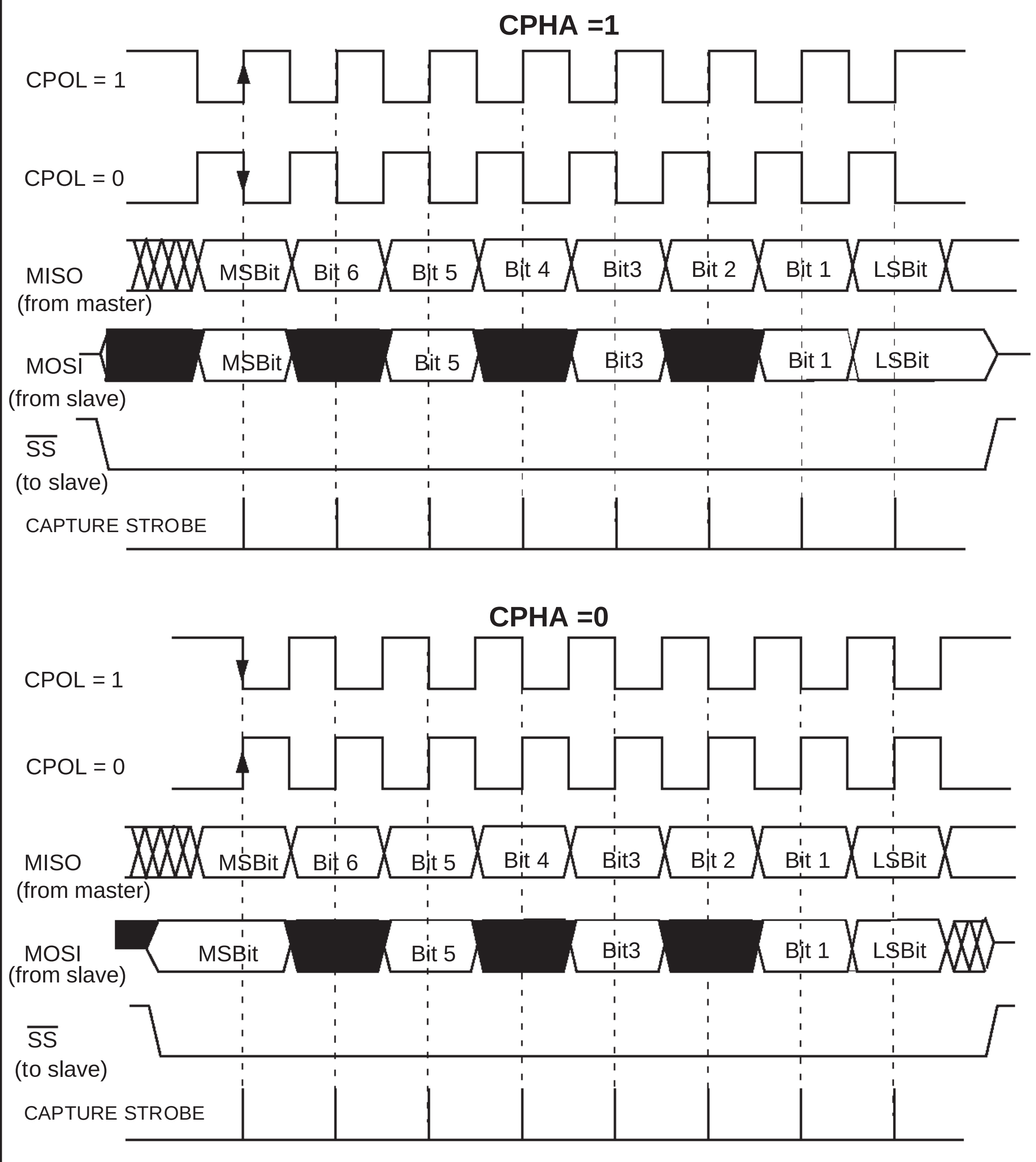
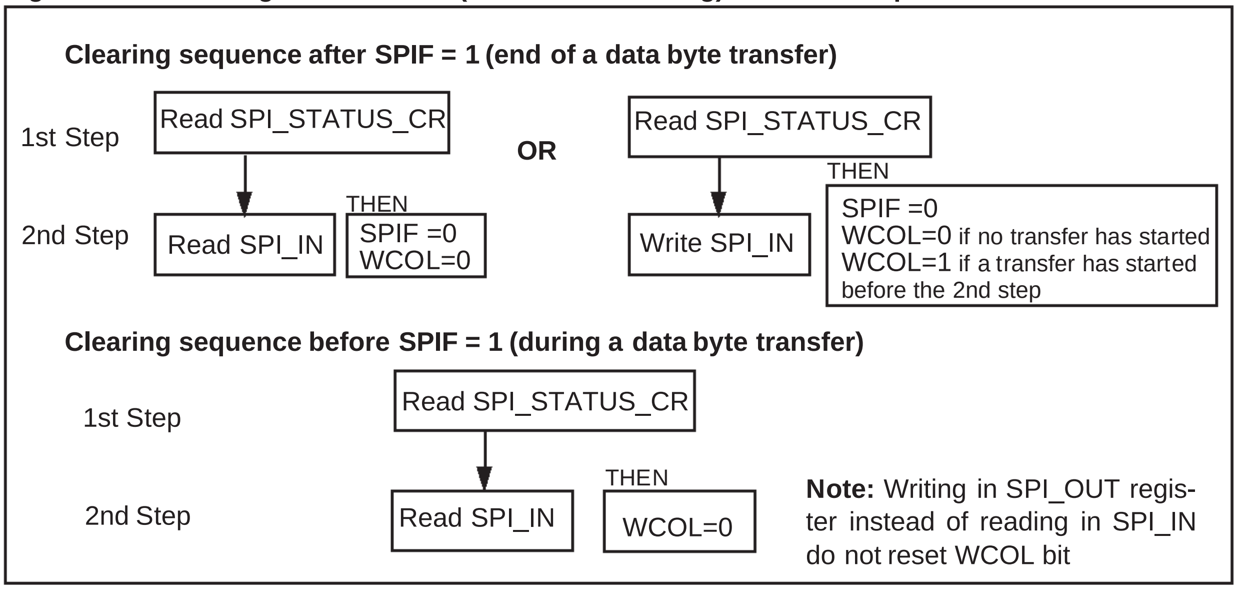


Figure 13.4   Data Clock Timing Diagram



Note: This figure should not be used as a replacement for parametric information. Refer to the Electrical Characteristics chapter.

Figure 13.5 Clearing the WCOL bit (Write Collision Flag) Software Sequence


Clearing sequence after SPIF = 1 (end of a data byte transfer)
1st Step
2nd Step
Read SPI_STATUS_CR
Read SPI_IN
THEN
SPIF =0
WCOL=0
OR
Read SPI_STATUS_CR
Write SPI_IN
THEN
SPIF =0
WCOL=0 if no transfer has started
WCOL=1 if a transfer has started before the 2nd step
Clearing sequence before SPIF = 1 (during a data byte transfer)
1st Step
2nd Step
Read SPI_STATUS_CR
Read SPI_IN
THEN
WCOL=0
Note: Writing in SPI_OUT register instead of reading in SPI_IN do not reset WCOL bit

13.4.6 Overrun Condition.

An overrun condition occurs when the master device has sent several data bytes and the slave device hasn't cleared the SPIF bit issued from the previous data byte transmitted.

In this case, the receiver buffer contains the byte sent after the SPIF bit was last cleared. A read to the SPI_IN register returns this byte. All other bytes are lost.

This condition is not detected by the SPI peripheral.

13.4.7 Single Master and Multimaster Configurations.

There are two types of SPI systems:

– Single Master System

– Multimaster System

Single Master System

A typical single master system may be configured, using an ICU as the master and four ICUs as slaves (see Figure 13.6).

The master device selects the individual slave devices by using four pins of a parallel port to control the four $\overline{SS}$ pins of the slave devices.

The $\overline{SS}$ pins are pulled high during reset since the master device ports will be forced to be inputs at that time, thus disabling the slave devices.

Note: *In order to prevent a bus conflict on the MISO line the master allows only one active slave device during a transmission.*

For more security, the slave device may respond to the master with the data byte received. Then the master will receive the previous byte back from the slave device if all MISO and MOSI pins are connected and the slave has not written its SPI_OUT register.

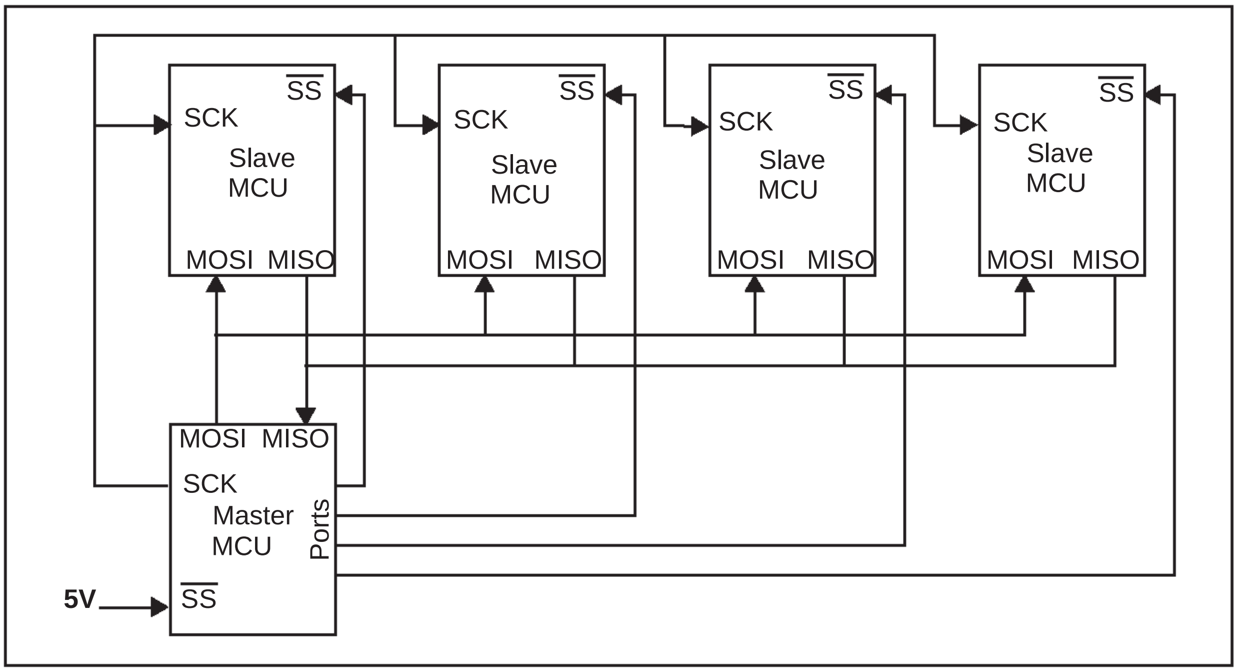
Other transmission security methods can use ports for handshake lines or data bytes with command fields.

Multi-master System

A multi-master system may also be configured by the user. Transfer of master control could be implemented using a handshake method through the I/O ports or by an exchange of code messages through the serial peripheral interface system.

The multi-master system is principally handled by the MSTR bit in the SPI_CR register and the MODF bit in the SPI_STATUS_CR register.

Figure 13.6 Single Master Configuration



13.4.8 Interrupts

Interrupt Event	Event Flag	Enable Control Bit	Exit from Wait	Exit from Halt
SPI End of Transfer Event	SPIF	SPIE	Yes	No
Master Mode Fault Event	MODF		Yes	No

Note: *The SPI interrupt events are connected to the same interrupt vector (see Interrupts chapter). They generate an interrupt if the corresponding Enable Control Bit (SPIE) and the interrupt mask bit (MSKSPI) in the INT_MASK Configuration Register is set.*

13.5 SPI Register Description

In the following sections describe the registers used by the SPI. In the 16 pin devices the SPI is not present and the described register aren't used

13.5.1 SPI Configuration Registers.

SPI Control Register (SPI_CR)

Configuration Register 20 (014h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
SPIE	SPE	SPR2	MSTR	CPOL	CPHA	SPR1	SPR2

Bit 7: **SPIE** Serial peripheral interrupt enable.

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An SPI interrupt is generated whenever SPIF=1 or MODF=1 in SPI_STATUS_CR

Bit 6: **SPE** Serial peripheral output enable.

This bit is set and cleared by software. It is also cleared by hardware when, in master mode, $\overline{SS}$=0 (see Section 13.4.5 Master Mode Fault).

0: I/O port connected to pins

1: SPI alternate functions connected to pins

Note: *The SPE bit is cleared by reset, so the SPI peripheral is not initially connected to the pins.*

Bit 5: **SPR2** Divider Enable.

This bit is set and cleared by software and it is cleared by reset. It is used with the SPR[1:0] bits to set the baud rate. Refer to Table 13.1.

0: Divider by 2 enabled

1: Divider by 2 disabled

Note: *This bit has no effect in slave mode.*

Bit 4: **MSTR** Master/Slave mode select.

This bit is set and cleared by software. It is also cleared by hardware when, in master mode, $\overline{SS}$=0 (see Section 13.4.5 Master Mode Fault).

0: Slave mode is selected

1: Master mode is selected, the function of the SCK pin changes from an input to an output and the functions of the MISO and MOSI pins are reversed.

Bit 3: **CPOL** Clock polarity.

This bit is set and cleared by software. This bit determines the steady state of the serial Clock. The CPOL bit affects both the master and slave modes.

0: The steady state is a low value at the SCK pin.

1: The steady state is a high value at the SCK pin.

Note: *SPI must be disabled by resetting the SPE bit if CPOL is changed at the communication byte boundaries.*

Bit 2: **CPHA** Clock phase.

This bit is set and cleared by software.

0: The first clock transition is the first data capture edge.

1: The second clock transition is the first capture edge.

Bit 1-0: **SPR1-SPR0** Serial peripheral rate.

These bits are set and cleared by software. Used with the SPR2 bit, they select one of six baud rates to be used as the serial clock when the device is a master (see Table 13.1).

These 2 bits have no effect in slave mode.

Remark: *It is recommended to write the SPI_CR register after the SPI_STATUS_CR register.*

Table 13.1 Serial Peripheral Baud Rate

Serial Clock	SPR2	SPR1	SPR0
$f_{CPU}/2$	1	0	0
$f_{CPU}/4$	0	0	0
$f_{CPU}/8$	0	0	1
$f_{CPU}/16$	1	1	0
$f_{CPU}/32$	0	1	0
$f_{CPU}/64$	0	1	1

SPI Control-Status Register (SPI_STATUS_CR)

Configuration Register 21 (015h) Read/Write

Reset Value: 0000 0000 (00h)

7							0
SPIE	SPE	SPR2	MSTR	CPOL	CPHA	SPR1	SPR2

Bit 7: **SPIF** Serial Peripheral data transfer flag. (read only)

This bit is set by hardware when a transfer has been completed. An interrupt is generated if SPIE=1 in the SPI_CR register. It is cleared by a software sequence (an access to the SPI_STATUS_CR register followed by a read or write to the SPI_IN/ SPI_OUT registers).

0: Data transfer is in progress or has been approved by a clearing sequence.

1: Data transfer between the device and an external device has been completed.

Note: *While the SPIF bit is set, all writes to the SPI_OUT register are inhibited.*

Bit 6: **WCOL** Write Collision status (read only).

This bit is set by hardware when a write to the SPI_OUT register is done during a transmit sequence. It is cleared by a software sequence (see Figure 13.5).

0: No write collision occurred

1: A write collision has been detected

Bit 5: **OR** SPI overrun error (read only).

This bit is set by hardware when the byte currently being received in the shift register is ready to be transferred into the SPI_IN register while SPIF = 1 (See Section 13.4.6 Overrun Condition). An interrupt is generated if SPIE = 1 in SPI_CR register. It is cleared by a software sequence (read of the SPI_STATUS_CR register followed by a read in SPI_IN or write of the SPI_OUT register).

0: No overrun error.

1: Overrun error detected.

Bit 4: **MODF** Mode Fault flag (read only).

This bit is set by hardware when the $\overline{SS}$ pin is pulled low in master mode (see Section 13.4.5 Master Mode Fault). An SPI interrupt can be generated if SPIE=1 in the SPI_CR register. This bit is cleared by a software sequence (An access to the SPI_STATUS_CR register while MODF=1 followed by a write to the SPI_CR register).

0: No master mode fault detected

1: A fault in master mode has been detected

Bit 3: Not used.

Bit 2: **SOD** SPI output disable

This bit is set and cleared by software. When set, it disables the alternate function of the SPI output (MOSI in master mode / MISO in slave mode)

0: SPI output not disable

1: SPI output disable.

Bit 1: **SSM** $\overline{SS}$ mode selection

This bit is set and cleared by software. When set, it disables the alternate function of the SPI Slave Select pin and use the SSI bit value instead of.

0: $\overline{SS}$ pin used by the SPI.

1: $\overline{SS}$ pin not used (I/O mode), SSI bit value is used.

Bit 0: **SSI** $\overline{SS}$ internal mode

This bit is set and cleared by software. It replaces pin $\overline{SS}$ of the SPI when bit SSM is set to 1. SSI bit is active low slave select signal when SSM is set to 1.

0 : Slave selected

1 : Slave not selected.

Remark: *It is recommended to write the SPI_STATUS_CR register before the SPI_CR register.*

13.5.2 SPI Input Register.

SPI Data Input Register (SPI_IN)

Input Register 5 (05h) Read only

Reset Value: 0000 0000 (00h)

7							0
SPIDI7	SPIDI6	SPIDI5	SPIDI4	SPIDI3	SPIDI2	SPIDI1	SPIDI0

bit 7-0: **SPIDI7-SPIDI0** Received data.

The SPI_IN register is used to receive data on the serial bus.

Note: *During the last clock cycle the SPIF bit is set, a copy of the data byte received in the shift register is moved to a buffer. When the user reads the serial peripheral data I/O register, the buffer is actually being read.*

Warning: *A read to the SPI_IN register returns the value located in the buffer and not the contents of the shift register (see Figure 13.2).*

13.5.3 SPI Output Register.

SPI Data Output Register (SPI_OUT)

Output Register 5 (05h) Read only

Reset Value: 0000 0000 (00h)

7							0
SPIDO7	SPIDO6	SPIDO5	SPIDO4	SPIDO3	SPIDO2	SPIDO1	SPIDO0

bit 7-0: **SPIDO7-SPIDO0** Data to be transmitted.

The SPI_OUT register is used to transmit data on the serial bus. In the master device only a write to this register will initiate transmission/reception of another byte.

Warning: *A write to the SPI_OUT register places data directly into the shift register for transmission.*