



---

---

## Section 34. Controller Area Network (CAN)

---

---

### HIGHLIGHT

This section of the manual contains the following topics:

34.1	Introduction .....	34-2
34.2	CAN Message Formats .....	34-4
34.3	CAN Registers .....	34-9
34.4	Enabling and Disabling the CAN Module .....	34-47
34.5	CAN Module Operating Modes .....	34-47
34.6	CAN Message Handling .....	34-49
34.7	Transmitting a CAN Message .....	34-56
34.8	CAN Message Filtering .....	34-68
34.9	Receiving a CAN Message .....	34-75
34.10	Bit Timing .....	34-83
34.11	CAN Error Management .....	34-86
34.12	CAN Interrupts .....	34-89
34.13	CAN Received Message Time Stamping .....	34-93
34.14	Low-Power Modes .....	34-93
34.15	Related Application Notes .....	34-95
34.16	Revision History .....	34-96

**Note:** This family reference manual section is meant to serve as a complement to device data sheets. Depending on the device variant, this manual section may not apply to all PIC32 devices.

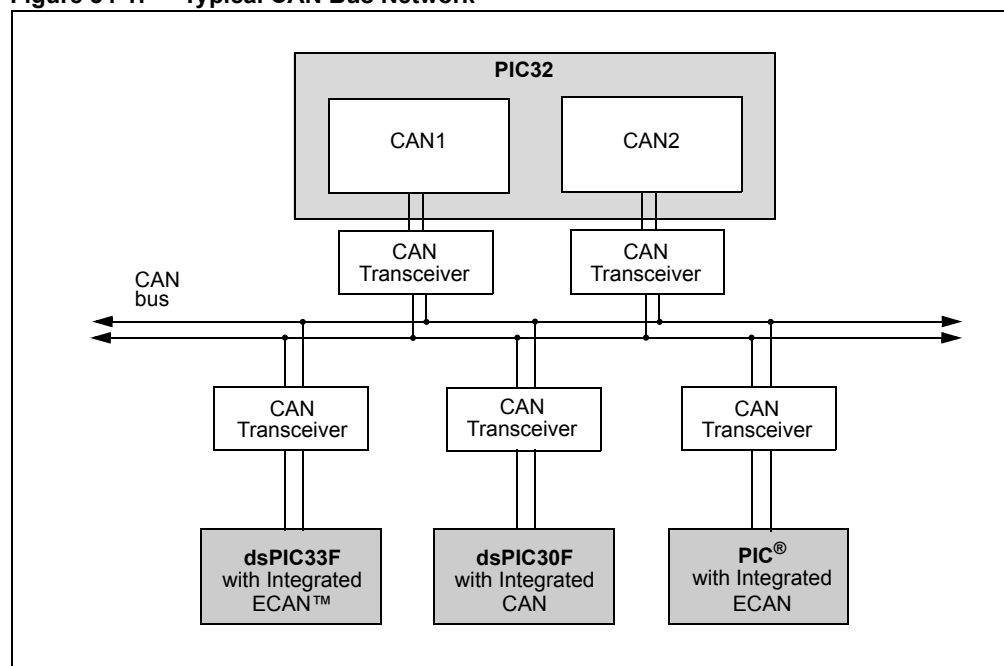
Please consult the note at the beginning of the “**Controller Area Network (CAN)**” chapter in the current device data sheet to check whether this document supports the device you are using.

Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Web site at: <http://www.microchip.com>

## 34.1 INTRODUCTION

The PIC32 Controller Area Network (CAN) module implements the CAN Specification 2.0B, which is used primarily in industrial and automotive applications. This asynchronous serial data communication protocol provides reliable communication in an electrically noisy environment. The PIC32 device family integrates up to two CAN modules. Figure 34-1 illustrates a typical CAN bus topology.

**Figure 34-1: Typical CAN Bus Network**



The CAN module supports the following key features:

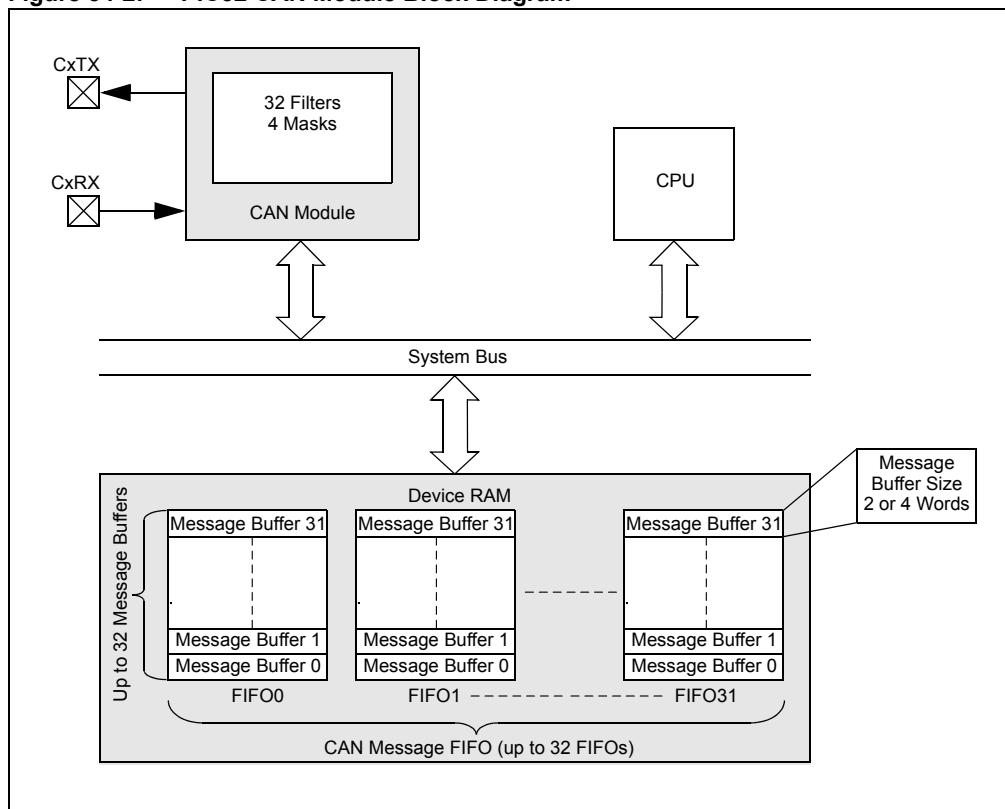
- Standards Compliance:
  - Full CAN Specification 2.0B compliance
  - Programmable bit rate up to 1 Mbps
- Message Reception and Transmission:
  - 32 message FIFOs
  - Each FIFO can have up to 32 messages for a total of 1024 messages
  - FIFO can be a transmit message FIFO or a receive message FIFO
  - User-defined priority levels for message FIFOs used for transmission
  - 32 acceptance filters for message filtering
  - Four acceptance filter mask registers for message filtering
  - Automatic response to Remote Transmit Request (RTR)
  - DeviceNet™ addressing support

## Section 34. Controller Area Network (CAN)

- Additional Features:
  - Loopback, Listen All Messages and Listen-Only modes for self-test, system diagnostics and bus monitoring
  - Low-power operating modes
  - CAN module is a bus master on the PIC32 system bus
  - Does not require Direct Memory Access (DMA) channels for operation
  - Dedicated time stamp timer
  - Data-only Message Reception mode

Figure 34-2 illustrates the general structure of the CAN module.

**Figure 34-2: PIC32 CAN Module Block Diagram**



The CAN module consists of a protocol engine, message acceptance filters and Message Assembly Buffers (MABs). The protocol engine transmits and receives messages to and from the CAN bus (as per CAN Specification 2.0B). Received messages are assembled in the receive message assembly buffer. The received message is then filtered by the message acceptance filters. The transmit message assembly buffer holds the message to be transmitted as it is processed by the protocol engine.

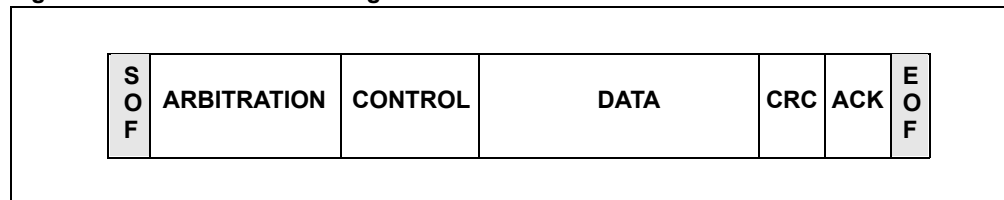
The CAN message buffers reside in device RAM. There are no CAN message buffers in the CAN module. Therefore, all messages are stored in device RAM. The CAN module is a bus master on the PIC32 system bus, and will read and write data to device RAM as required. The CAN module does not use DMA for its operation and fetches messages from the device RAM without DMA or CPU intervention.

## 34.2 CAN MESSAGE FORMATS

The CAN bus protocol uses asynchronous communication. Information is passed from the transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame as illustrated in [Figure 34-3](#).

Each frame begins with a Start of Frame (SOF) bit field and terminates with an End of Frame (EOF) bit field. The SOF is followed by the Arbitration and Control fields, which identify the message type, format, length and priority. This information allows each node on the CAN bus to respond appropriately to the message. The Data field conveys the message content and is of variable length, ranging from 0 bytes to 8 bytes. Error protection is provided by the Cyclic Redundancy Check (CRC) and Acknowledgement (ACK) fields.

**Figure 34-3: CAN Bus Message Frame**



The CAN bus protocol supports four frame types:

- **Data Frame** – carries data from transmitter to the receivers
- **Remote Frame** – transmitted by a node on the bus, to request transmission of a data frame with the same identifier from another node
- **Error Frame** – transmitted by any node when it detects an error
- **Overload Frame** – provides an extra delay between successive Data or remote frames
- **Interframe Space** – provides a separation between successive frames

The CAN Specification 2.0B defines two additional data formats:

- **Standard Data Frame** – intended for standard messages that use 11 identifier bits
- **Extended Data Frame** – intended for extended messages that use 29 identifier bits

There are three CAN Specification versions:

- **2.0A** – considers 29-bit identifier as error
- **2.0B Passive** – ignores 29-bit identifier messages
- **2.0B Active** – handles both 11-bit and 29-bit identifiers

The PIC32 CAN module is compliant with the CAN Specification 2.0B, while providing enhanced message filtering capabilities.

**Note:** For detailed information on the CAN protocol, refer to the Bosch CAN Bus Specification.

# Section 34. Controller Area Network (CAN)

## 34.2.1 Standard Data Frame

The standard data frame message begins with an SOF bit followed by a 12-bit Arbitration field as illustrated in Figure 34-4. The Arbitration field contains an 11-bit identifier and RTR bit. The identifier defines the type of information contained in the message, and is used by each receiving node to determine if the message is of interest to it. The RTR bit distinguishes a data frame from a remote frame. For a standard data frame, the RTR bit is clear.

Following the Arbitration field is a 6-bit Control field, which provides more information about the contents of the message. The first bit in the Control field is an Identifier Extension (IDE) bit, which distinguishes the message as either a standard or extended data frame. A standard data frame is indicated by a dominant state (logic level '0') during transmission of the IDE bit. The second bit in the Control field is a reserved (RB0) bit, which is in the dominant state (logic level '0'). The last four bits in the Control field represent the Data Length Code (DLC), which specifies the number of data bytes present in the message.

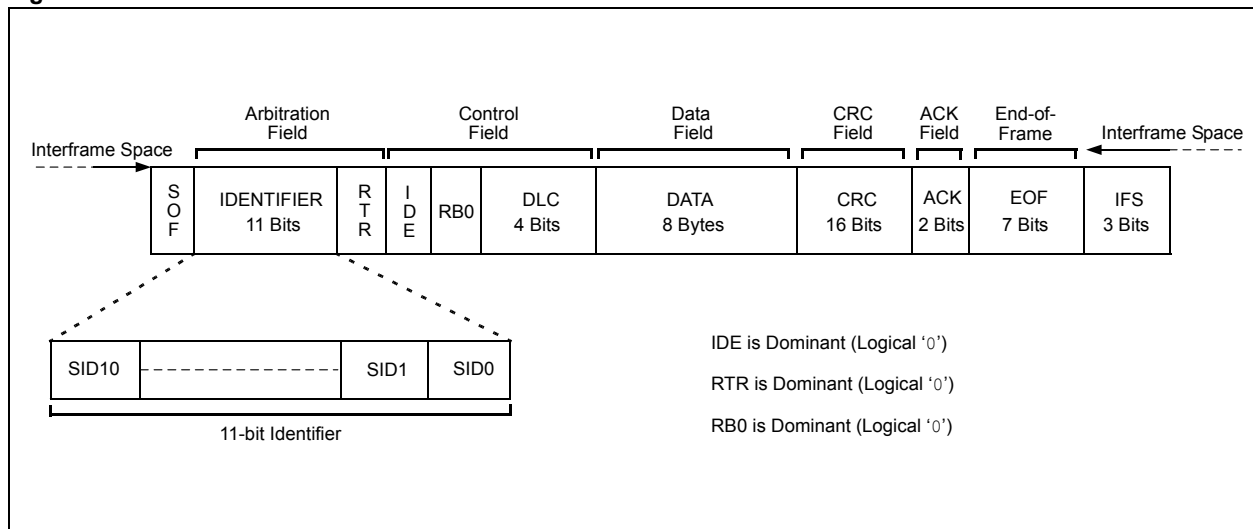
The Data field follows the Control field. This field carries the message data – the actual payload of the data frame. This field is of variable length, ranging from 0 bytes to eight bytes. The number of bytes is user-selectable.

The Data field is followed by the CRC field, which is a 15-bit CRC sequence with one delimiter bit.

The Acknowledgement (ACK) field is sent as a recessive bit (logic level '1'), and is overwritten as a dominant bit by any receiver that has received the data correctly. The message is acknowledged by the receiver regardless of the result of the acceptance filter comparison.

The last field is the EOF field, which consists of seven recessive bits that indicate the end of message.

Figure 34-4: Format of the Standard Data Frame



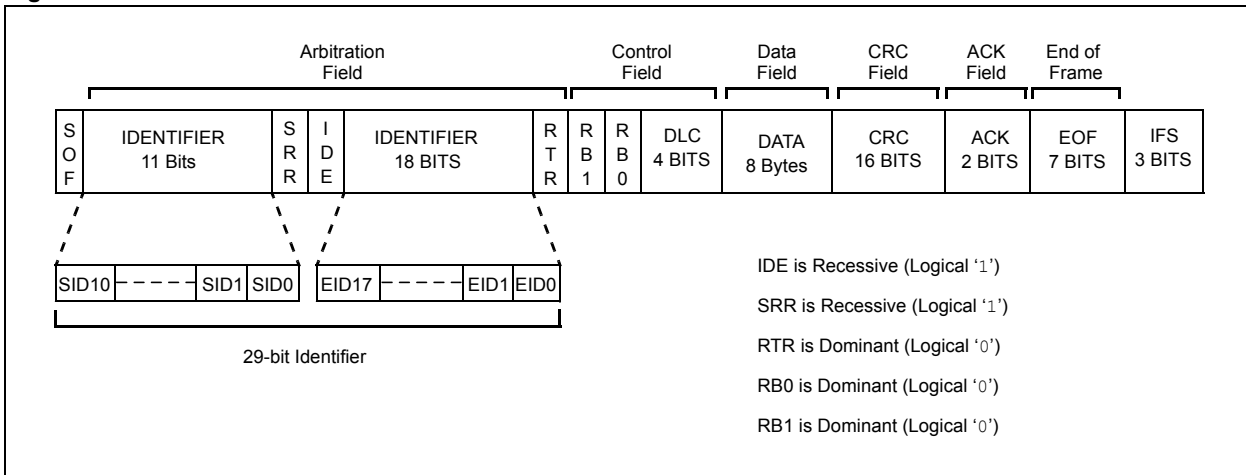
## 34.2.2 Extended Data Frame

The extended data frame begins with an SOF bit followed by a 31-bit Arbitration field as illustrated in Figure 34-5. The Arbitration field for the extended data frame contains 29 identifier bits in two fields separated by a Substitute Remote Request (SRR) bit and an IDE bit. The SRR bit determines if the message is a remote frame. SRR = 1 for extended data frames. The IDE bit indicates the data frame type. For the extended data frame, IDE = 1.

The extended data frame Control field consists of seven bits. The first bit is the RTR. For the extended data frame, RTR = 0. The next two bits, RB1 and RB0, are reserved bits that are in the dominant state (logic level '0'). The last four bits in the Control field are the DLC, which specifies the number of data bytes present in the message.

The remaining fields in an extended data frame are identical to a standard data frame.

**Figure 34-5: Format of the Extended Data Frame**



# Section 34. Controller Area Network (CAN)

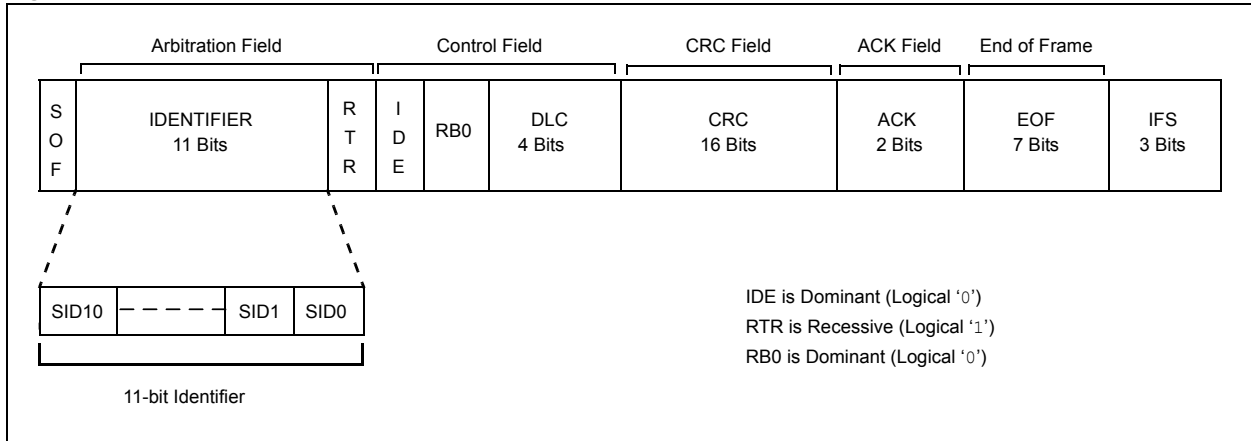
## 34.2.3 Remote Frame

A node expecting to receive data from another node can initiate transmission of the respective data by the source node, by sending a remote frame. A remote frame can be in standard format (Figure 34-6) or the extended format (Figure 34-7).

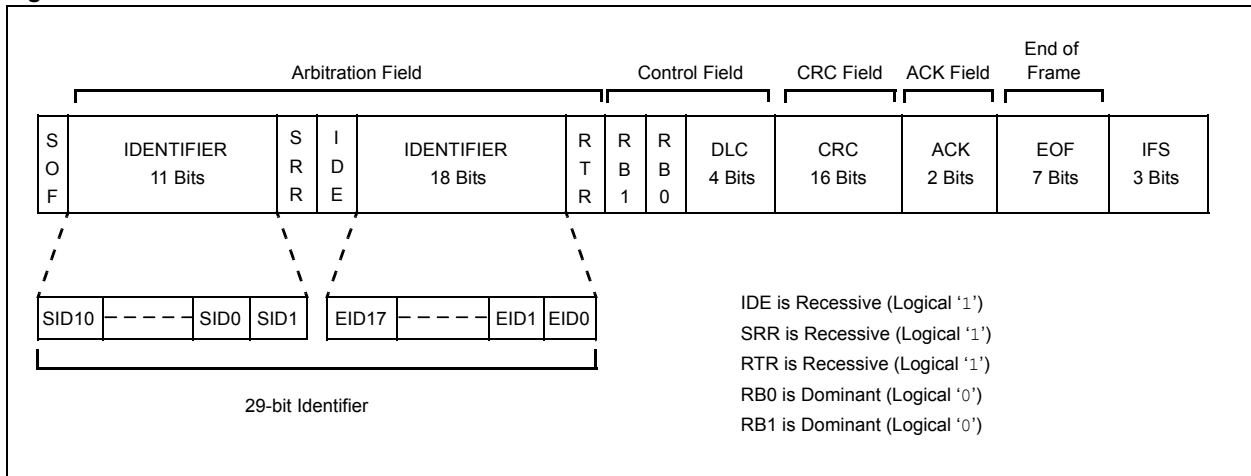
A Remote frame is similar to a data frame, with the following exceptions:

- The RTR bit is recessive (RTR = 1)
- There is no Data field (DLC = 0)

**Figure 34-6: Format of the Standard Remote Frame**



**Figure 34-7: Format of the Extended Remote Frame**



## 34.2.4 Error Frame

An error frame is generated by any node that detects a bus error. An error frame consists of an Error Flag field followed by an Error Delimiter field. The Error Delimiter consists of eight recessive bits and allows the bus nodes to restart communication cleanly after an error has occurred. There are two types of error flag fields, depending on the error status of the node that detects the error:

- **Error Active Flag** – contains six consecutive dominant bits, which forces all other nodes on the network to generate Error Echo Fags, thereby resulting in a series of 6 to 12 dominant bits on the bus
- **Error Passive Flag** – contains six consecutive recessive bits, with the result that unless the bus error is detected by the transmitting node, the transmission of an Error Passive Flag will not affect the communication of any other node on the network

## 34.2.5 Overload Frame

An Overload Frame can be generated by a node either when a dominant bit is detected during Interframe Space or when a node is not ready to receive the next message (for example, if it is still reading the previous received message). An Overload Frame has the same format as an Error Frame with an Active Error Flag, but can only be generated during Interframe Space. It consists of an Overload Fag field with six dominant bits followed by an Overload Delimiter field with eight recessive bits. A node can generate a maximum of two sequential overload frames to delay the start of the next message.

## 34.2.6 Interframe Space

Interframe Space separates successive frames being transmitted on the CAN bus. It consists of at least three recessive bits, referred to as intermission. The Interframe Space allows nodes time to internally process the previously received message before the start of the next frame. If the transmitting node is in the Error Passive state, an additional eight recessive bits will be inserted in the Interframe Space before any other message is transmitted by the node. This period is called a Suspend Transmit field and allows time for other transmitting nodes to take control of the bus.



## 34.3 CAN REGISTERS

The CAN module registers can be classified by their function into the following groups:

- Module and CAN bit rate Configuration registers
- Interrupt and Status registers
- Mask and Filter Configuration registers
- FIFO Control registers

### 34.3.1 Module and CAN Bit Rate Configuration Registers

**Note:** The 'i' shown in the register identifier denotes CAN1 or CAN2.

- **CiCON: CAN Module Control Register**

This register is used to set up the CAN module operational mode and DeviceNet addressing.

- **CiCFG: CAN Baud Rate Configuration Register<sup>(1)</sup>**

This register contains control bits to set the period of each time quantum, using the baud rate prescaler, and specifies Synchronization Jump Width (SJW) in terms of time quanta. It is also used to program the number of time quanta in each CAN bit segment, including the propagation and phase segments 1 and 2.

### 34.3.2 Interrupt and Status Registers

- **CiINT: CAN Interrupt Register**

This register allows various CAN module interrupt sources to be enabled and disabled. It also contains interrupt status flags.

- **CiVEC: CAN Interrupt Code Register**

This register provides status bits which provide information on CAN module interrupt source and message filter hits. These values can be used to implement a jump table for handling different cases.

- **CiTREC: CAN Transmit/Receive Error Count Register**

This register provides information on Transmit and Receive Error Counter values. It also has bits which indicate various warning states.

- **CiFSTAT: CAN FIFO Status Register**

This register contains interrupt status flag for all the FIFOs.

- **CiRXOVF: CAN Receive FIFO Overflow Status Register**

This register contains overflow interrupt status flag for all the FIFOs.

- **CiTMR: CAN Timer Register**

This register contains CAN Message Timestamp timer and a Prescaler.

### 34.3.3 Mask and Filter Configuration Registers

- **CiRXMn: CAN Acceptance Filter Mask n Register (n = 0, 1, 2 or 3)<sup>(1)</sup>**

These registers allow the configuration of the filter masks. A total of four masks are available.

- **CiFLTCON0: CAN Filter Control Register 0<sup>(1)</sup>**

These registers allow the association of FIFO and Masks with a filter. A Filter can be associated with any one mask. It also contains a filter enable/disable bit.

- **CiRXFn: CAN Acceptance Filter n Register 7 (n = 0 through 31)<sup>(1)</sup>**

These registers specify the filter to be applied to the received message. A total of 32 filters are available.

## 34.3.4 CAN Module Control Registers

- **CiFIFOBA: CAN Message Buffer Base Address Register<sup>(1)</sup>**

This register holds the base (start) address of the CAN message buffer area. This is a physical address.

- **CiFIFOCONn: CAN FIFO Control Register (n = 0 through 31)**

These registers allow the control and configuration of CAN Message FIFOs.

- **CiFIFOINTn: CAN FIFO Interrupt Register (n = 0 through 31)**

These registers allow the individual FIFO interrupt sources to be enabled or disabled. They also contain interrupt status bits.

- **CiFIFOUAn: CAN FIFO User Address Register (n = 0 through 31)<sup>(1)</sup>**

These registers provide the address of the memory location in the CAN message FIFO from where the next message can be read or where the next message should be written to.

- **CiFIFOCIn: CAN Module Message Index Register (n = 0 through 31)**

These registers provide the message buffer index (in the message FIFO) of the next message that the CAN module will transmit or where the next received message will be saved.

[Table 34-1](#) provides a summary of all CAN-related registers. Corresponding registers appear after the summary, followed by a detailed description of each register. All unimplemented registers and/or bits within a register read as zeros.

# Section 34. Controller Area Network (CAN)

**Table 34-1: CAN Controller Register Summary**

Address Offset	Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0		
0x00	CiCON <sup>(1,2,3)</sup>	31:24	—	—	—	—	ABAT	REQOP<2:0>				
		23:16	OPMOD<2:0>				CANCAP	—	—	—	—	
		15:8	ON	—	SIDLE	—	CANBUSY	—	—	—		
		7:0	—	—	—	DNCNT<4:0>						
0x10	CiCFG <sup>(1,2,3)</sup>	31:24	—	—	—	—	—	—	—	—		
		23:16	—	WAKFIL	—	—	—	SEG2PH<2:0>				
		15:8	SEG2PHTS	SAM	SEG1PH<2:0>			PRSEG<2:0>				
		7:0	SJW<1:0>		BRP<5:0>							
0x20	CiINT <sup>(1,2,3)</sup>	31:24	IVRIE	WAKIE	CERRIE	SERRIE	RBOVIE	—	—	—		
		23:16	—	—	—	—	MODIE	CTMRIE	RBIE	TBIE		
		15:8	IVRIF	WAKIF	CERRIF	SERRIF	RBOVIF	—	—	—		
		7:0	—	—	—	—	MODIF	CTMRIF	RBIF	TBIF		
0x30	CiVEC <sup>(1,2,3)</sup>	31:24	—	—	—	—	—	—	—	—		
		23:16	—	—	—	—	—	—	—	—		
		15:8	—	—	—	FILHIT<4:0>						
		7:0	ICODE<6:0>									
0x40	CiTREC <sup>(1,2,3)</sup>	31:24	—	—	—	—	—	—	—	—		
		23:16	—	—	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN		
		15:8	TERRCNT<7:0>									
		7:0	RERRCNT<7:0>									
0x50	CiFSTAT <sup>(1,2,3)</sup>	31:24	FIFOIP31	FIFOIP30	FIFOIP29	FIFOIP28	FIFOIP27	FIFOIP26	FIFOIP25	FIFOIP24		
		23:16	FIFOIP23	FIFOIP22	FIFOIP21	FIFOIP20	FIFOIP19	FIFOIP18	FIFOIP17	FIFOIP16		
		15:8	FIFOIP15	FIFOIP14	FIFOIP13	FIFOIP12	FIFOIP11	FIFOIP10	FIFOIP9	FIFOIP8		
		7:0	FIFOIP7	FIFOIP6	FIFOIP5	FIFOIP4	FIFOIP3	FIFOIP2	FIFOIP1	FIFOIP0		
0x60	CiRXOVF <sup>(1,2,3)</sup>	31:24	RXOVF31	RXOVF30	RXOVF29	RXOVF28	RXOVF27	RXOVF26	RXOVF25	RXOVF24		
		23:16	RXOVF23	RXOVF22	RXOVF21	RXOVF20	RXOVF19	RXOVF18	RXOVF17	RXOVF16		
		15:8	RXOVF15	RXOVF14	RXOVF13	RXOVF12	RXOVF11	RXOVF10	RXOVF9	RXOVF8		
		7:0	RXOVF7	RXOVF6	RXOVF5	RXOVF4	RXOVF3	RXOVF2	RXOVF1	RXOVF0		
0x70	CiTMR <sup>(1,2,3)</sup>	31:24	CANTS<15:8>									
		23:16	CANTS<7:0>									
		15:8	CANTSPRE<15:8>									
		7:0	CANTSPRE<7:0>									
0x80	CiRXM0 <sup>(1,2,3)</sup>	31:24	SID<10:3>									
		23:16	SID<2:0>			—	MIDE	—	EID<17:16>			
		15:8	EID<15:8>									
		7:0	EID<7:0>									
0x90	CiRXM1 <sup>(1,2,3)</sup>	31:24	SID<10:3>									
		23:16	SID<2:0>			—	MIDE	—	EID<17:16>			
		15:8	EID<15:8>									
		7:0	EID<7:0>									
0xA0	CiRXM2 <sup>(1,2,3)</sup>	31:24	SID<10:3>									
		23:16	SID<2:0>			—	MIDE	—	EID<17:16>			
		15:8	EID<15:8>									
		7:0	EID<7:0>									

- Note 1:** This register has an associated Clear register at an offset of 0x4 bytes. These registers have the same name with CLR appended to the end of the register name (For example, CiCONCLR). Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.
- Note 2:** This register has an associated Set register at an offset of 0x8 bytes. These registers have the same name with SET appended to the end of the register name (For example, CiCONSET). Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.
- Note 3:** This register has an associated Invert register at an offset of 0xC bytes. These registers have the same name with INV appended to the end of the register name (For example, CiCONINV). Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

# PIC32 Family Reference Manual

**Table 34-1: CAN Controller Register Summary (Continued)**

Address Offset	Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0	
0xB0	CiRXM3 <sup>(1,2,3)</sup>	31:24	SID<10:3>								
		23:16	SID<2:0>			—	MIDE	—	EID<17:16>		
		15:8	EID<15:8>								
		7:0	EID<7:0>								
0xC0	CiFLTCO0 <sup>(1,2,3)</sup>	31:24	FLTEN3	MSEL3<1:0>			FSEL3<4:0>				
		23:16	FLTEN2	MSEL2<1:0>			FSEL2<4:0>				
		15:8	FLTEN1	MSEL1<1:0>			FSEL1<4:0>				
		7:0	FLTEN0	MSEL0<1:0>			FSEL0<4:0>				
0xD0	CiFLTCO1 <sup>(1,2,3)</sup>	31:24	FLTEN7	MSEL7<1:0>			FSEL7<4:0>				
		23:16	FLTEN6	MSEL6<1:0>			FSEL6<4:0>				
		15:8	FLTEN5	MSEL5<1:0>			FSEL5<4:0>				
		7:0	FLTEN4	MSEL4<1:0>			FSEL4<4:0>				
0xE0	CiFLTCO2 <sup>(1,2,3)</sup>	31:24	FLTEN11	MSEL11<1:0>			FSEL11<4:0>				
		23:16	FLTEN10	MSEL10<1:0>			FSEL10<4:0>				
		15:8	FLTEN9	MSEL9<1:0>			FSEL9<4:0>				
		7:0	FLTEN8	MSEL8<1:0>			FSEL8<4:0>				
0xF0	CiFLTCO3 <sup>(1,2,3)</sup>	31:24	FLTEN15	MSEL15<1:0>			FSEL15<4:0>				
		23:16	FLTEN14	MSEL14<1:0>			FSEL14<4:0>				
		15:8	FLTEN13	MSEL13<1:0>			FSEL13<4:0>				
		7:0	FLTEN12	MSEL12<1:0>			FSEL12<4:0>				
0x100	CiFLTCO4 <sup>(1,2,3)</sup>	31:24	FLTEN19	MSEL19<1:0>			FSEL19<4:0>				
		23:16	FLTEN18	MSEL18<1:0>			FSEL18<4:0>				
		15:8	FLTEN17	MSEL17<1:0>			FSEL17<4:0>				
		7:0	FLTEN16	MSEL16<1:0>			FSEL16<4:0>				
0x110	CiFLTCO5 <sup>(1,2,3)</sup>	31:24	FLTEN23	MSEL23<1:0>			FSEL23<4:0>				
		23:16	FLTEN22	MSEL22<1:0>			FSEL22<4:0>				
		15:8	FLTEN21	MSEL21<1:0>			FSEL21<4:0>				
		7:0	FLTEN20	MSEL20<1:0>			FSEL20<4:0>				
0x120	CiFLTCO6 <sup>(1,2,3)</sup>	31:24	FLTEN27	MSEL27<1:0>			FSEL27<4:0>				
		23:16	FLTEN26	MSEL26<1:0>			FSEL26<4:0>				
		15:8	FLTEN25	MSEL25<1:0>			FSEL25<4:0>				
		7:0	FLTEN24	MSEL24<1:0>			FSEL24<4:0>				
0x130	CiFLTCO7 <sup>(1,2,3)</sup>	31:24	FLTEN31	MSEL31<1:0>			FSEL31<4:0>				
		23:16	FLTEN30	MSEL30<1:0>			FSEL30<4:0>				
		15:8	FLTEN29	MSEL29<1:0>			FSEL29<4:0>				
		7:0	FLTEN28	MSEL28<1:0>			FSEL28<4:0>				
0x140	CiRXFn <sup>(1,2,3)</sup> (n = 0 through 31)	31:24	SID<10:3>								
		23:16	SID<2:0>			—	EXID	—	EID<17:16>		
		15:8	EID<15:8>								
		7:0	EID<7:0>								
0x340	CiFIFOBA <sup>(1,2,3)</sup>	31:24	CiFIFOBA<31:24>								
		23:16	CiFIFOBA<23:16>								
		15:8	CiFIFOBA<15:8>								
		7:0	CiFIFOBA<7:0>								
0x350	CiFIFO- CONn <sup>(1,2,3)</sup> (n = 0 through 31)	31:24	—	—	—	—	—	—	—	—	
		23:16	—	—	—	FSIZE<4:0>					
		15:8	—	FRESET	UINC	DONLY	—	—	—	—	
		7:0	TXEN	TXABAT	TXLARB	TXERR	TXREQ	RTREN	TXPR<1:0>		

- Note 1:** This register has an associated Clear register at an offset of 0x4 bytes. These registers have the same name with CLR appended to the end of the register name (For example, CiCONCLR). Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.
- Note 2:** This register has an associated Set register at an offset of 0x8 bytes. These registers have the same name with SET appended to the end of the register name (For example, CiCONSET). Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.
- Note 3:** This register has an associated Invert register at an offset of 0xC bytes. These registers have the same name with INV appended to the end of the register name (For example, CiCONINV). Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

## Section 34. Controller Area Network (CAN)

**Table 34-1: CAN Controller Register Summary (Continued)**

Address Offset	Name	Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0	
0x360	CiFIFOINT <sub>n</sub> <sup>(1,2,3)</sup> (n = 0 through 31)	31:24	—	—	—	—	—	TXNFULLIE	TXHALFIE	TXEMPTYIE	
		23:16	—	—	—	—	RXOVFLIE	RXFULLIE	RXHALFIE	RXEMPTYIE	
		15:8	—	—	—	—	—	TXNFULLIF	TXHALFIF	TXEMPTYIF	
		7:0	—	—	—	—	RXOVFLIF	RXFULLIF	RXHALFIF	RXEMPTYIF	
0x370	CiFIFOUA <sub>n</sub> <sup>(1,2,3)</sup> (n = 0 through 31)	31:24	CiFIFOUA<31:24>								
		23:16	CiFIFOUA<23:16>								
		15:8	CiFIFOUA<15:8>								
		7:0	CiFIFOUA<7:0>								
0x380	CiFIFOCI <sub>n</sub> <sup>(1,2,3)</sup> (n = 0 through 31)	31:24	—	—	—	—	—	—	—	—	
		23:16	—	—	—	—	—	—	—	—	
		15:8	—	—	—	—	—	—	—	—	
		7:0	—	—	—	CiFIFOCI<4:0>					

- Note 1:** This register has an associated Clear register at an offset of 0x4 bytes. These registers have the same name with CLR appended to the end of the register name (For example, CiCONCLR). Writing a '1' to any bit position in the Clear register will clear valid bits in the associated register. Reads from the Clear register should be ignored.
- 2:** This register has an associated Set register at an offset of 0x8 bytes. These registers have the same name with SET appended to the end of the register name (For example, CiCONSET). Writing a '1' to any bit position in the Set register will set valid bits in the associated register. Reads from the Set register should be ignored.
- 3:** This register has an associated Invert register at an offset of 0xC bytes. These registers have the same name with INV appended to the end of the register name (For example, CiCONINV). Writing a '1' to any bit position in the Invert register will invert valid bits in the associated register. Reads from the Invert register should be ignored.

# PIC32 Family Reference Manual

**Register 34-1: CiCON: CAN Module Control Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	S/HC-0	R/W-1	R/W-0	R/W-0
	—	—	—	—	ABAT	REQOP<2:0>		
23:16	R-1	R-0	R-0	R/W-0	U-0	U-0	U-0	U-0
	OPMOD<2:0>			CANCAP	—	—	—	—
15:8	R/W-0	U-0	R/W-0	U-0	R-0	U-0	U-0	U-0
	ON <sup>(1)</sup>	—	SIDL	—	CANBUSY	—	—	—
7:0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	DNCNT<4:0>				

**Legend:** HC = Hardware Clear S = Settable bit  
R = Readable bit W = Writable bit P = Programmable bit r = Reserved bit  
U = Unimplemented bit -n = Bit Value at POR: ('0', '1', x = Unknown)

bit 31-28 **Unimplemented:** Read as '0'

bit 27 **ABAT:** Abort All Pending Transmissions bit  
1 = Signal all transmit buffers to abort transmission  
0 = Module will clear this bit when all transmissions aborted

bit 26-24 **REQOP<2:0>:** Request Operation Mode bits  
111 = Set Listen All Messages mode  
110 = Reserved - Do not use  
101 = Reserved - Do not use  
100 = Set Configuration mode  
011 = Set Listen Only mode  
010 = Set Loopback mode  
001 = Set Disable mode  
000 = Set Normal Operation mode

bit 23-21 **OPMOD<2:0>:** Operation Mode Status bits  
111 = Module is in Listen All Messages mode  
110 = Reserved  
101 = Reserved  
100 = Module is in Configuration mode  
011 = Module is in Listen Only mode  
010 = Module is in Loopback mode  
001 = Module is in Disable mode  
000 = Module is in Normal Operation mode

bit 20 **CANCAP:** CAN Message Receive Time Stamp Timer Capture Enable bit  
1 = CANTMR value is stored on valid message reception and is stored with the message  
0 = Disable CAN message receive time stamp timer capture and stop CANTMR to conserve power

bit 19-16 **Unimplemented:** Read as '0'

bit 15 **ON:** CAN On bit<sup>(1)</sup>  
1 = CAN module is enabled  
0 = CAN module is disabled

bit 14 **Unimplemented:** Read as '0'

**Note 1:** If the user application clears this bit it may take a number of cycles before the CAN module completes the current transaction and responds to this request. The user application should poll the CANBUSY bit to verify that the request has been honored.

## Section 34. Controller Area Network (CAN)

---

### Register 34-1: CiCON: CAN Module Control Register (Continued)

- bit 13 **SIDLE:** CAN Stop in Idle bit  
1 = CAN Stops operation when system enters Idle mode  
0 = CAN continues operation when system enters Idle mode
- bit 12 **Unimplemented:** Read as '0'
- bit 11 **CANBUSY:** CAN Module is Busy bit  
1 = The CAN module is active  
0 = The CAN module is completely disabled
- bit 10-5 **Unimplemented:** Read as '0'
- bit 4-0 **DNCNT<4:0>:** Device Net Filter Bit Number bits  
10011-11111 = Invalid Selection (compare up to 18-bits of data with EID)  
10010 = Compare up to data byte 2 bit 6 with EID17 (CiRXFn<17>)  
•  
•  
•  
00001 = Compare up to data byte 0 bit 7 with EID0 (CiRXFn<0>)  
00000 = Do not compare data bytes

**Note 1:** If the user application clears this bit it may take a number of cycles before the CAN module completes the current transaction and responds to this request. The user application should poll the CANBUSY bit to verify that the request has been honored.

# PIC32 Family Reference Manual

**Register 34-2: CiCFG: CAN Baud Rate Configuration Register<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
	—	WAKFIL	—	—	—	SEG2PH<2:0> <sup>(2,5)</sup>		
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SEG2PHTS <sup>(2)</sup>	SAM <sup>(3)</sup>	SEG1PH<2:0>			PRSEG<2:0>		
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SJW<1:0> <sup>(4)</sup>		BRP<5:0>					

**Legend:** HC = Hardware Clear S = Settable bit  
R = Readable bit W = Writable bit P = Programmable bit r = Reserved bit  
U = Unimplemented bit -n = Bit Value at POR: ('0', '1', x = Unknown)

bit 31-23 **Unimplemented:** Read as '0'

bit 22 **WAKFIL:** CAN Bus Line Filter Enable bit  
1 = Use CAN bus line filter for wake-up  
0 = CAN bus line filter is not used for wake-up

bit 21-19 **Unimplemented:** Read as '0'

bit 18-16 **SEG2PH<2:0>:** Phase Buffer Segment 2 bits<sup>(2,5)</sup>  
111 = Length is 8 x T<sub>Q</sub>  
•  
•  
•  
000 = Length is 1 x T<sub>Q</sub>

bit 15 **SEG2PHTS:** Phase Segment 2 Time Select bit<sup>(2)</sup>  
1 = Freely programmable  
0 = Maximum of SEG1PH or Information Processing Time, whichever is greater

bit 14 **SAM:** Sample of the CAN Bus Line bit<sup>(3)</sup>  
1 = Bus line is sampled three times at the sample point  
0 = Bus line is sampled once at the sample point

bit 13-11 **SEG1PH<2:0>:** Phase Buffer Segment 1 bits<sup>(5)</sup>  
111 = Length is 8 x T<sub>Q</sub>  
•  
•  
•  
000 = Length is 1 x T<sub>Q</sub>

**Note 1:** This register can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> (CiCON<23:21>) = 100).

- 2:** SEG2PH ≤ SEG1PH. If SEG2PHTS is clear, SEG2PH will be set automatically.
- 3:** 3 Time bit sampling is not allowed for BRP < 2.
- 4:** SJW ≤ SEG2PH.
- 5:** The Time Quanta per bit must be greater than 7 (that is, T<sub>QBIT</sub> > 7).



## Section 34. Controller Area Network (CAN)

### Register 34-2: CiCFG: CAN Baud Rate Configuration Register<sup>(1)</sup> (Continued)

bit 10-8 **PRSEG<2:0>**: Propagation Time Segment bits<sup>(5)</sup>

111 = Length is 8 x T<sub>Q</sub>

•  
•  
•

000 = Length is 1 x T<sub>Q</sub>

bit 7-6 **SJW<1:0>**: Synchronization Jump Width bits<sup>(4)</sup>

11 = Length is 4 x T<sub>Q</sub>

10 = Length is 3 x T<sub>Q</sub>

01 = Length is 2 x T<sub>Q</sub>

00 = Length is 1 x T<sub>Q</sub>

bit 5-0 **BRP<5:0>**: Baud Rate Prescaler bits

111111 = T<sub>Q</sub> = (2 x 64)/F<sub>SYS</sub>

111110 = T<sub>Q</sub> = (2 x 63)/F<sub>SYS</sub>

•  
•  
•

000001 = T<sub>Q</sub> = (2 x 2)/F<sub>SYS</sub>

000000 = T<sub>Q</sub> = (2 x 1)/F<sub>SYS</sub>

**Note 1:** This register can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> (CiCON<23:21>) = 100).

**2:** SEG2PH ≤ SEG1PH. If SEG2PHTS is clear, SEG2PH will be set automatically.

**3:** 3 Time bit sampling is not allowed for BRP < 2.

**4:** SJW ≤ SEG2PH.

**5:** The Time Quanta per bit must be greater than 7 (that is, T<sub>QBIT</sub> > 7).

# PIC32 Family Reference Manual

**Register 34-3: CIINT: CAN Interrupt Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
	IVRIE	WAKIE	CERRIE	SERRIE	RBOVIE	—	—	—
23:16	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	—	MODIE	CTMRIE	RBIE	TBIE
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0
	IVRIF	WAKIF	CERRIF	SERRIF <sup>(1)</sup>	RBOVIF	—	—	—
7:0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	—	MODIF	CTMRIF	RBIF	TBIF

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31      **IVRIE:** Invalid Message Received Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 30      **WAKIE:** CAN Bus Activity Wake-up Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 29      **CERRIE:** CAN Bus Error Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 28      **SERRIE:** System Error Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 27      **RBOVIE:** Receive Buffer Overflow Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 26-20 **Unimplemented:** Read as '0'
- bit 19      **MODIE:** Mode Change Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 18      **CTMRIE:** CAN Timestamp Timer Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 17      **RBIE:** Receive Buffer Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 16      **TBIE:** Transmit Buffer Interrupt Enable bit  
             1 = Interrupt request enabled  
             0 = Interrupt request not enabled
- bit 15      **IVRIF:** Invalid Message Received Interrupt Flag bit  
             1 = An invalid messages interrupt has occurred  
             0 = An invalid message interrupt has not occurred

**Note 1:** This bit can only be cleared by turning the CAN module Off and On by clearing or setting the ON bit (CiCON<15>).

## Section 34. Controller Area Network (CAN)

### Register 34-3: CiINT: CAN Interrupt Register (Continued)

- bit 14 **WAKIF:** CAN Bus Activity Wake-up Interrupt Flag bit  
1 = A bus wake-up activity interrupt has occurred  
0 = A bus wake-up activity interrupt has not occurred
- bit 13 **CERRIF:** CAN Bus Error Interrupt Flag bit  
1 = A CAN bus error has occurred  
0 = A CAN bus error has not occurred
- bit 12 **SERRIF:** System Error Interrupt Flag bit  
1 = A system error occurred (typically an illegal address was presented to the system bus)  
0 = A system error has not occurred
- bit 11 **RBOVIF:** Receive Buffer Overflow Interrupt Flag bit  
1 = A receive buffer overflow has occurred  
0 = A receive buffer overflow has not occurred
- bit 10-4 **Unimplemented:** Read as '0'
- bit 3 **MODIF:** CAN Mode Change Interrupt Flag bit  
1 = A CAN module mode change has occurred (OPMOD<2:0> has changed to reflect REQOP)  
0 = A CAN module mode change has not occurred
- bit 2 **CTMRIF:** CAN Timer Overflow Interrupt Flag bit  
1 = A CAN timer (CANTMR) overflow has occurred  
0 = A CAN timer (CANTMR) overflow has not occurred
- bit 1 **RBIF:** Receive Buffer Interrupt Flag bit  
1 = A receive buffer interrupt is pending  
0 = A receive buffer interrupt is not pending
- bit 0 **TBIF:** Transmit Buffer Interrupt Flag bit  
1 = A transmit buffer interrupt is pending  
0 = A transmit buffer interrupt is not pending

**Note 1:** This bit can only be cleared by turning the CAN module Off and On by clearing or setting the ON bit (CiCON<15>).

# PIC32 Family Reference Manual

**Register 34-4: CIVEC: CAN Interrupt Code Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
15:8	U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
	—	—	—	FILHIT<4:0>				
7:0	U-0	R-1	R-0	R-0	R-0	R-0	R-0	R-0
	—	ICODE<6:0> <sup>(1)</sup>						

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-13 **Unimplemented:** Read as '0'

bit 12-8 **FILHIT<4:0>:** Filter Hit Number bit

11111 = Filter 31

11110 = Filter 30

•

•

•

00001 = Filter 1

00000 = Filter 0

bit 7 **Unimplemented:** Read as '0'

bit 6-0 **ICODE<6:0>:** Interrupt Flag Code bits<sup>(1)</sup>

1001000-1111111 = Reserved

1001000 = Invalid message received (IVRIF)

1000111 = CAN module mode change (MODIF)

1000110 = CAN timestamp timer (CTMRIF)

1000101 = Bus bandwidth error (SERRIF)

1000100 = Address error interrupt (SERRIF)

1000011 = Receive FIFO overflow interrupt (RBOVIF)

1000010 = Wake-up interrupt (WAKIF)

1000001 = Error Interrupt (CERRIF)

1000000 = No interrupt

0100000-0111111 = Reserved

0011111 = FIFO31 Interrupt (CiFSTAT<31> set)

0011110 = FIFO30 Interrupt (CiFSTAT<30> set)

•

•

•

0000001 = FIFO1 Interrupt (CiFSTAT<1> set)

0000000 = FIFO0 Interrupt (CiFSTAT<0> set)

**Note 1:** These bits are only updated for enabled interrupts.

## Section 34. Controller Area Network (CAN)

**Register 34-5: CiTREC: CAN Transmit/Receive Error Count Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
	—	—	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
15:8	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	TERRCNT<7:0>							
7:0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RERRCNT<7:0>							

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31-22 **Unimplemented:** Read as '0'
- bit 21 **TXBO:** Transmitter in Error State Bus OFF (TERRCNT ≥ 256)
- bit 20 **TXBP:** Transmitter in Error State Bus Passive (TERRCNT ≥ 128)
- bit 19 **RXBP:** Receiver in Error State Bus Passive (RERRCNT ≥ 128)
- bit 18 **TXWARN:** Transmitter in Error State Warning (128 > TERRCNT ≥ 96)
- bit 17 **RXWARN:** Receiver in Error State Warning (128 > RERRCNT ≥ 96)
- bit 16 **EWARN:** Transmitter or Receiver is in Error State Warning
- bit 15-8 **TERRCNT<7:0>:** Transmit Error Counter
- bit 7-0 **RERRCNT<7:0>:** Receive Error Counter

**Register 34-6: CiFSTAT: CAN FIFO Status Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	FIFOIP31	FIFOIP30	FIFOIP29	FIFOIP28	FIFOIP27	FIFOIP26	FIFOIP25	FIFOIP24
23:16	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	FIFOIP23	FIFOIP22	FIFOIP21	FIFOIP20	FIFOIP19	FIFOIP18	FIFOIP17	FIFOIP16
15:8	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	FIFOIP15	FIFOIP14	FIFOIP13	FIFOIP12	FIFOIP11	FIFOIP10	FIFOIP9	FIFOIP8
7:0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	FIFOIP7	FIFOIP6	FIFOIP5	FIFOIP4	FIFOIP3	FIFOIP2	FIFOIP1	FIFOIP0

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31-0 **FIFOIP<31:0>:** FIFO Interrupt Pending bits
  - 1 = One or more enabled FIFO interrupts are pending
  - 0 = No FIFO interrupts are pending

# PIC32 Family Reference Manual

**Register 34-7: CiRXOVF: CAN Receive FIFO Overflow Status Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RXOVF31	RXOVF30	RXOVF29	RXOVF28	RXOVF27	RXOVF26	RXOVF25	RXOVF24
23:16	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RXOVF23	RXOVF22	RXOVF21	RXOVF20	RXOVF19	RXOVF18	RXOVF17	RXOVF16
15:8	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RXOVF15	RXOVF14	RXOVF13	RXOVF12	RXOVF11	RXOVF10	RXOVF9	RXOVF8
7:0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
	RXOVF7	RXOVF6	RXOVF5	RXOVF4	RXOVF3	RXOVF2	RXOVF1	RXOVF0

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-0 **RXOVF<31:0>**: FIFO Receive Overflow Interrupt Pending bit

- 1 = FIFO has overflowed
- 0 = FIFO has not overflowed

**Register 34-8: CiTMR: CAN Timer Register**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	CANTS<15:8>							
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	CANTS<7:0>							
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	CANTSPRE<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	CANTSPRE<7:0>							

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-0 **CANTS<15:0>**: CAN Time Stamp Timer bits

This is a free-running timer that increments every CANTSPRE system clocks when the CANCAP bit (CiCON<20>) is set.

bit 15-0 **CANTSPRE<15:0>**: CAN Time Stamp Timer Prescaler bits

65535 = CAN time stamp timer (CANTS) increments every 65,535 system clocks

- 
- 
- 

0 = CAN time stamp timer (CANTS) increments every system clock

**Note 1:** CiTMR will be frozen when CANCAP = 0.

**2:** The CiTMR prescaler count will be reset on any write to CiTMR (CANTSPRE will be unaffected).

## Section 34. Controller Area Network (CAN)

**Register 34-9: CiRXMn: CAN Acceptance Filter Mask n Register (n = 0, 1, 2 or 3)<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	SID<10:3>							
23:16	R/W-0	R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0	R/W-0
	SID<2:0>			—	MIDE	—	EID<17:16>	
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	EID<15:8>							
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	EID<7:0>							

**Legend:**

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

- bit 31-21 **SID<10:0>**: Standard Identifier bits
  - 1 = Include bit, SIDx, in filter comparison
  - 0 = Bit SIDx is 'don't care' in filter operation
- bit 20 **Unimplemented**: Read as '0'
- bit 19 **MIDE**: Identifier Receive Mode bit
  - 1 = Match only message types (standard/extended address) that correspond to the EXID bit in filter
  - 0 = Match either standard or extended address message if filters match (that is, if (Filter SID) = (Message SID) or if (FILTER SID/EID) = (Message SID/EID))
- bit 18 **Unimplemented**: Read as '0'
- bit 17-0 **EID<17:0>**: Extended Identifier bits
  - 1 = Include bit, EIDx, in filter comparison
  - 0 = Bit EIDx is 'don't care' in filter operation

**Note 1:** This register can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> (CiCON<23:21>) = 100).

# PIC32 Family Reference Manual

**Register 34-10: CiFLTCON0: CAN Filter Control Register 0<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN3	MSEL3<1:0>		FSEL3<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN2	MSEL2<1:0>		FSEL2<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN1	MSEL1<1:0>		FSEL1<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN0	MSEL0<1:0>		FSEL0<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31      **FLTEN3:** Filter 3 Enable bit  
             1 = Filter is enabled  
             0 = Filter is disabled
- bit 30-29      **MSEL3<1:0>:** Filter 3 Mask Select bits  
             11 = Acceptance Mask 3 selected  
             10 = Acceptance Mask 2 selected  
             01 = Acceptance Mask 1 selected  
             00 = Acceptance Mask 0 selected
- bit 28-24      **FSEL3<4:0>:** FIFO Selection bits  
             11111 = Message matching filter is stored in FIFO buffer 31  
             11110 = Message matching filter is stored in FIFO buffer 30  
             .  
             .  
             .  
             00001 = Message matching filter is stored in FIFO buffer 1  
             00000 = Message matching filter is stored in FIFO buffer 0
- bit 23      **FLTEN2:** Filter 2 Enable bit  
             1 = Filter is enabled  
             0 = Filter is disabled
- bit 22-21      **MSEL2<1:0>:** Filter 2 Mask Select bits  
             11 = Acceptance Mask 3 selected  
             10 = Acceptance Mask 2 selected  
             01 = Acceptance Mask 1 selected  
             00 = Acceptance Mask 0 selected
- bit 20-16      **FSEL2<4:0>:** FIFO Selection bits  
             11111 = Message matching filter is stored in FIFO buffer 31  
             11110 = Message matching filter is stored in FIFO buffer 30  
             .  
             .  
             .  
             00001 = Message matching filter is stored in FIFO buffer 1  
             00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.



## Section 34. Controller Area Network (CAN)

### Register 34-10: CiFLTCON0: CAN Filter Control Register 0<sup>(1)</sup> (Continued)

bit 15	<b>FLTEN1:</b> Filter 1 Enable bit 1 = Filter is enabled 0 = Filter is disabled
bit 14-13	<b>MSEL1&lt;1:0&gt;:</b> Filter 1 Mask Select bits 11 = Acceptance Mask 3 selected 10 = Acceptance Mask 2 selected 01 = Acceptance Mask 1 selected 00 = Acceptance Mask 0 selected
bit 12-8	<b>FSEL1&lt;4:0&gt;:</b> FIFO Selection bits 11111 = Message matching filter is stored in FIFO buffer 31 11110 = Message matching filter is stored in FIFO buffer 30 • • • 00001 = Message matching filter is stored in FIFO buffer 1 00000 = Message matching filter is stored in FIFO buffer 0
bit 7	<b>FLTEN0:</b> Filter 0 Enable bit 1 = Filter is enabled 0 = Filter is disabled
bit 6-5	<b>MSEL0&lt;1:0&gt;:</b> Filter 0 Mask Select bits 11 = Acceptance Mask 3 selected 10 = Acceptance Mask 2 selected 01 = Acceptance Mask 1 selected 00 = Acceptance Mask 0 selected
bit 4-0	<b>FSEL0&lt;4:0&gt;:</b> FIFO Selection bits 11111 = Message matching filter is stored in FIFO buffer 31 11110 = Message matching filter is stored in FIFO buffer 30 • • • 00001 = Message matching filter is stored in FIFO buffer 1 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-11: CiFLTCON1: CAN Filter Control Register 1<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN7	MSEL7<1:0>		FSEL7<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN6	MSEL6<1:0>		FSEL6<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN5	MSEL5<1:0>		FSEL5<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN4	MSEL4<1:0>		FSEL4<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31      **FLTEN7**: Filter 7 Enable bit

1 = Filter is enabled  
 0 = Filter is disabled

bit 30-29 **MSEL7<1:0>**: Filter 7 Mask Select bits

11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected

bit 28-24 **FSEL7<4:0>**: FIFO Selection bits

11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

bit 23      **FLTEN6**: Filter 6 Enable bit

1 = Filter is enabled  
 0 = Filter is disabled

bit 22-21 **MSEL6<1:0>**: Filter 6 Mask Select bits

11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected

bit 20-16 **FSEL6<4:0>**: FIFO Selection bits

11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-11: CiFLTCON1: CAN Filter Control Register 1<sup>(1)</sup> (Continued)

- bit 15 **FLTEN5**: Filter 5 Enable bit  
1 = Filter is enabled  
0 = Filter is disabled
- bit 14-13 **MSEL5<1:0>**: Filter 5 Mask Select bits  
11 = Acceptance Mask 3 selected  
10 = Acceptance Mask 2 selected  
01 = Acceptance Mask 1 selected  
00 = Acceptance Mask 0 selected
- bit 12-8 **FSEL5<4:0>**: FIFO Selection bits  
11111 = Message matching filter is stored in FIFO buffer 31  
11110 = Message matching filter is stored in FIFO buffer 30  
•  
•  
•  
00001 = Message matching filter is stored in FIFO buffer 1  
00000 = Message matching filter is stored in FIFO buffer 0
- bit 7 **FLTEN4**: Filter 4 Enable bit  
1 = Filter is enabled  
0 = Filter is disabled
- bit 6-5 **MSEL4<1:0>**: Filter 4 Mask Select bits  
11 = Acceptance Mask 3 selected  
10 = Acceptance Mask 2 selected  
01 = Acceptance Mask 1 selected  
00 = Acceptance Mask 0 selected
- bit 4-0 **FSEL4<4:0>**: FIFO Selection bits  
11111 = Message matching filter is stored in FIFO buffer 31  
11110 = Message matching filter is stored in FIFO buffer 30  
•  
•  
•  
00001 = Message matching filter is stored in FIFO buffer 1  
00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-12: CiFLTCON2: CAN Filter Control Register 2<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN11	MSEL11<1:0>		FSEL11<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN10	MSEL10<1:0>		FSEL10<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN9	MSEL9<1:0>		FSEL9<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN8	MSEL8<1:0>		FSEL8<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31      **FLTEN11:** Filter 11 Enable bit

1 = Filter is enabled  
 0 = Filter is disabled

bit 30-29      **MSEL11<1:0>:** Filter 11 Mask Select bits

11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected

bit 28-24      **FSEL11<4:0>:** FIFO Selection bits

11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

bit 23      **FLTEN10:** Filter 10 Enable bit

1 = Filter is enabled  
 0 = Filter is disabled

bit 22-21      **MSEL10<1:0>:** Filter 10 Mask Select bits

11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected

bit 20-16      **FSEL10<4:0>:** FIFO Selection bits

11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-12: CiFLTCON2: CAN Filter Control Register 2<sup>(1)</sup> (Continued)

- bit 15     **FLTEN9**: Filter 9 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 14-13   **MSEL9<1:0>**: Filter 9 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 12-8    **FSEL9<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0
- bit 7       **FLTEN8**: Filter 8 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 6-5     **MSEL8<1:0>**: Filter 8 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 4-0     **FSEL8<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-13: CiFLTCON3: CAN Filter Control Register 3<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN15	MSEL15<1:0>		FSEL15<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN14	MSEL14<1:0>		FSEL14<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN13	MSEL13<1:0>		FSEL13<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN12	MSEL12<1:0>		FSEL12<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31      **FLTEN15:** Filter 15 Enable bit

- 1 = Filter is enabled
- 0 = Filter is disabled

bit 30-29      **MSEL15<1:0>:** Filter 15 Mask Select bits

- 11 = Acceptance Mask 3 selected
- 10 = Acceptance Mask 2 selected
- 01 = Acceptance Mask 1 selected
- 00 = Acceptance Mask 0 selected

bit 28-24      **FSEL15<4:0>:** FIFO Selection bits

- 11111 = Message matching filter is stored in FIFO buffer 31
- 11110 = Message matching filter is stored in FIFO buffer 30
- 
- 
- 
- 00001 = Message matching filter is stored in FIFO buffer 1
- 00000 = Message matching filter is stored in FIFO buffer 0

bit 23      **FLTEN14:** Filter 14 Enable bit

- 1 = Filter is enabled
- 0 = Filter is disabled

bit 22-21      **MSEL14<1:0>:** Filter 14 Mask Select bits

- 11 = Acceptance Mask 3 selected
- 10 = Acceptance Mask 2 selected
- 01 = Acceptance Mask 1 selected
- 00 = Acceptance Mask 0 selected

bit 20-16      **FSEL14<4:0>:** FIFO Selection bits

- 11111 = Message matching filter is stored in FIFO buffer 31
- 11110 = Message matching filter is stored in FIFO buffer 30
- 
- 
- 
- 00001 = Message matching filter is stored in FIFO buffer 1
- 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-13: CiFLTCON3: CAN Filter Control Register 3<sup>(1)</sup> (Continued)

- bit 15     **FLTEN13**: Filter 13 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 14-13   **MSEL13<1:0>**: Filter 13 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 12-8   **FSEL13<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0
- bit 7       **FLTEN12**: Filter 12 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 6-5     **MSEL12<1:0>**: Filter 12 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 4-0     **FSEL12<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-14: CiFLTCON4: CAN Filter Control Register 4<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN19	MSEL19<1:0>		FSEL19<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN18	MSEL18<1:0>		FSEL18<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN17	MSEL17<1:0>		FSEL17<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN16	MSEL16<1:0>		FSEL16<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31      **FLTEN19:** Filter 19 Enable bit  
 1 = Filter is enabled  
 0 = Filter is disabled
- bit 30-29      **MSEL19<1:0>:** Filter 19 Mask Select bits  
 11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected
- bit 28-24      **FSEL19<4:0>:** FIFO Selection bits  
 11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0
- bit 23      **FLTEN18:** Filter 18 Enable bit  
 1 = Filter is enabled  
 0 = Filter is disabled
- bit 22-21      **MSEL18<1:0>:** Filter 18 Mask Select bits  
 11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected
- bit 20-16      **FSEL18<4:0>:** FIFO Selection bits  
 11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 •  
 •  
 •  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.



## Section 34. Controller Area Network (CAN)

### Register 34-14: CiFLTCON4: CAN Filter Control Register 4<sup>(1)</sup> (Continued)

- bit 15     **FLTEN17**: Filter 17 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 14-13   **MSEL17<1:0>**: Filter 17 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 12-8    **FSEL17<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0
- bit 7       **FLTEN16**: Filter 16 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 6-5     **MSEL16<1:0>**: Filter 16 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 4-0     **FSEL16<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-15: CiFLTCON5: CAN Filter Control Register 5<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN23	MSEL23<1:0>		FSEL23<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN22	MSEL22<1:0>		FSEL22<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN21	MSEL21<1:0>		FSEL21<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN20	MSEL20<1:0>		FSEL20<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31      **FLTEN23:** Filter 23 Enable bit  
 1 = Filter is enabled  
 0 = Filter is disabled
- bit 30-29      **MSEL23<1:0>:** Filter 23 Mask Select bits  
 11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected
- bit 28-24      **FSEL23<4:0>:** FIFO Selection bits  
 11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 .  
 .  
 .  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0
- bit 23      **FLTEN22:** Filter 22 Enable bit  
 1 = Filter is enabled  
 0 = Filter is disabled
- bit 22-21      **MSEL22<1:0>:** Filter 22 Mask Select bits  
 11 = Acceptance Mask 3 selected  
 10 = Acceptance Mask 2 selected  
 01 = Acceptance Mask 1 selected  
 00 = Acceptance Mask 0 selected
- bit 20-16      **FSEL22<4:0>:** FIFO Selection bits  
 11111 = Message matching filter is stored in FIFO buffer 31  
 11110 = Message matching filter is stored in FIFO buffer 30  
 .  
 .  
 .  
 00001 = Message matching filter is stored in FIFO buffer 1  
 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-15: CiFLTCON5: CAN Filter Control Register 5<sup>(1)</sup> (Continued)

- bit 15     **FLTEN21**: Filter 21 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 14-13   **MSEL21<1:0>**: Filter 21 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 12-8   **FSEL21<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0
- bit 7       **FLTEN20**: Filter 20 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 6-5     **MSEL20<1:0>**: Filter 20 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 4-0     **FSEL20<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-16: CiFLTCON6: CAN Filter Control Register 6<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN27	MSEL27<1:0>		FSEL27<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN26	MSEL26<1:0>		FSEL26<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN25	MSEL25<1:0>		FSEL25<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN24	MSEL24<1:0>		FSEL24<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31      **FLTEN27:** Filter 27 Enable bit  
             1 = Filter is enabled  
             0 = Filter is disabled
- bit 30-29    **MSEL27<1:0>:** Filter 27 Mask Select bits  
             11 = Acceptance Mask 3 selected  
             10 = Acceptance Mask 2 selected  
             01 = Acceptance Mask 1 selected  
             00 = Acceptance Mask 0 selected
- bit 28-24    **FSEL27<4:0>:** FIFO Selection bits  
             11111 = Message matching filter is stored in FIFO buffer 31  
             11110 = Message matching filter is stored in FIFO buffer 30  
             •  
             •  
             •  
             00001 = Message matching filter is stored in FIFO buffer 1  
             00000 = Message matching filter is stored in FIFO buffer 0
- bit 23      **FLTEN26:** Filter 26 Enable bit  
             1 = Filter is enabled  
             0 = Filter is disabled
- bit 22-21    **MSEL26<1:0>:** Filter 26 Mask Select bits  
             11 = Acceptance Mask 3 selected  
             10 = Acceptance Mask 2 selected  
             01 = Acceptance Mask 1 selected  
             00 = Acceptance Mask 0 selected
- bit 20-16    **FSEL26<4:0>:** FIFO Selection bits  
             11111 = Message matching filter is stored in FIFO buffer 31  
             11110 = Message matching filter is stored in FIFO buffer 30  
             •  
             •  
             •  
             00001 = Message matching filter is stored in FIFO buffer 1  
             00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-16: CiFLTCON6: CAN Filter Control Register 6<sup>(1)</sup> (Continued)

- bit 15     **FLTEN25**: Filter 25 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 14-13 **MSEL25<1:0>**: Filter 25 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 12-8 **FSEL25<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0
- bit 7     **FLTEN24**: Filter 24 Enable bit  
          1 = Filter is enabled  
          0 = Filter is disabled
- bit 6-5 **MSEL24<1:0>**: Filter 24 Mask Select bits  
          11 = Acceptance Mask 3 selected  
          10 = Acceptance Mask 2 selected  
          01 = Acceptance Mask 1 selected  
          00 = Acceptance Mask 0 selected
- bit 4-0 **FSEL24<4:0>**: FIFO Selection bits  
          11111 = Message matching filter is stored in FIFO buffer 31  
          11110 = Message matching filter is stored in FIFO buffer 30  
          •  
          •  
          •  
          00001 = Message matching filter is stored in FIFO buffer 1  
          00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-17: CiFLTCON7: CAN Filter Control Register 7<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN31	MSEL31<1:0>		FSEL31<4:0>				
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN30	MSEL30<1:0>		FSEL30<4:0>				
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN29	MSEL29<1:0>		FSEL29<4:0>				
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	FLTEN28	MSEL28<1:0>		FSEL28<4:0>				

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31      **FLTEN31:** Filter 31 Enable bit

- 1 = Filter is enabled
- 0 = Filter is disabled

bit 30-29      **MSEL31<1:0>:** Filter 31 Mask Select bits

- 11 = Acceptance Mask 3 selected
- 10 = Acceptance Mask 2 selected
- 01 = Acceptance Mask 1 selected
- 00 = Acceptance Mask 0 selected

bit 28-24      **FSEL31<4:0>:** FIFO Selection bits

- 11111 = Message matching filter is stored in FIFO buffer 31
- 11110 = Message matching filter is stored in FIFO buffer 30
- .
- .
- .
- 00001 = Message matching filter is stored in FIFO buffer 1
- 00000 = Message matching filter is stored in FIFO buffer 0

bit 23      **FLTEN30:** Filter 30 Enable bit

- 1 = Filter is enabled
- 0 = Filter is disabled

bit 22-21      **MSEL30<1:0>:** Filter 30 Mask Select bits

- 11 = Acceptance Mask 3 selected
- 10 = Acceptance Mask 2 selected
- 01 = Acceptance Mask 1 selected
- 00 = Acceptance Mask 0 selected

bit 20-16      **FSEL30<4:0>:** FIFO Selection bits

- 11111 = Message matching filter is stored in FIFO buffer 31
- 11110 = Message matching filter is stored in FIFO buffer 30
- .
- .
- .
- 00001 = Message matching filter is stored in FIFO buffer 1
- 00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

## Section 34. Controller Area Network (CAN)

### Register 34-17: CiFLTCON7: CAN Filter Control Register 7<sup>(1)</sup> (Continued)

- bit 15 **FLTEN29**: Filter 29 Enable bit  
1 = Filter is enabled  
0 = Filter is disabled
- bit 14-13 **MSEL29<1:0>**: Filter 29 Mask Select bits  
11 = Acceptance Mask 3 selected  
10 = Acceptance Mask 2 selected  
01 = Acceptance Mask 1 selected  
00 = Acceptance Mask 0 selected
- bit 12-8 **FSEL29<4:0>**: FIFO Selection bits  
11111 = Message matching filter is stored in FIFO buffer 31  
11110 = Message matching filter is stored in FIFO buffer 30  
•  
•  
•  
00001 = Message matching filter is stored in FIFO buffer 1  
00000 = Message matching filter is stored in FIFO buffer 0
- bit 7 **FLTEN28**: Filter 28 Enable bit  
1 = Filter is enabled  
0 = Filter is disabled
- bit 6-5 **MSEL28<1:0>**: Filter 28 Mask Select bits  
11 = Acceptance Mask 3 selected  
10 = Acceptance Mask 2 selected  
01 = Acceptance Mask 1 selected  
00 = Acceptance Mask 0 selected
- bit 4-0 **FSEL28<4:0>**: FIFO Selection bits  
11111 = Message matching filter is stored in FIFO buffer 31  
11110 = Message matching filter is stored in FIFO buffer 30  
•  
•  
•  
00001 = Message matching filter is stored in FIFO buffer 1  
00000 = Message matching filter is stored in FIFO buffer 0

**Note 1:** The bits in this register can only be modified if the corresponding filter enable (FLTENn) bit is '0'.

# PIC32 Family Reference Manual

**Register 34-18: CiRXFn: CAN Acceptance Filter n Register 7 (n = 0 through 31)<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID<10:3>								
23:16	R/W-x	R/W-x	R/W-x	U-0	R/W-0	U-0	R/W-x	R/W-x
SID<2:0>			—	EXID	—	EID<17:16>		
15:8	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID<15:8>								
7:0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID<7:0>								

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

- bit 31-21 **SID<10:0>**: Standard Identifier bits
  - 1 = Message address bit SIDx must be '1' to match filter
  - 0 = Message address bit SIDx must be '0' to match filter
- bit 20 **Unimplemented**: Read as '0'
- bit 19 **EXID**: Extended Identifier Enable bits
  - 1 = Match only messages with extended identifier addresses
  - 0 = Match only messages with standard identifier addresses
- bit 18 **Unimplemented**: Read as '0'
- bit 17-0 **EID<17:0>**: Extended Identifier bits
  - 1 = Message address bit EIDx must be '1' to match filter
  - 0 = Message address bit EIDx must be '0' to match filter

**Note 1:** This register can only be modified when the filter is disabled (FLTENn = 0).



## Section 34. Controller Area Network (CAN)

**Register 34-19: CiFIFOBA: CAN Message Buffer Base Address Register<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CiFIFOBA<31:24>								
23:16	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CiFIFOBA<23:16>								
15:8	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CiFIFOBA<15:8>								
7:0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0 <sup>(2)</sup>	R-0 <sup>(2)</sup>
CiFIFOBA<7:0>								

**Legend:**

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared      x = Bit is unknown

bit 31-0 **CiFIFOBA<31:0>**: CAN FIFO Base Address bits

Defines the base address of all message buffers. Individual message buffers are located based on the size of the previous message buffers. This address is a physical address. Note that bits <1:0> are read-only and read '0', forcing the messages to be 32-bit word-aligned in device RAM.

**Note 1:** This register can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> (CiCON<23:21>) = 100).

**2:** This bit is unimplemented and will always read '0', which forces word-alignment of messages.).

# PIC32 Family Reference Manual

**Register 34-20: CiFIFOCONn: CAN FIFO Control Register (n = 0 through 31)**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
	—	—	—	—	—	—	—	—
23:16	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	FSIZE<4:0> <sup>(1)</sup>				
15:8	U-0	S/HC-0	S/HC-0	R/W-0	U-0	U-0	U-0	U-0
	—	FRESET	UINC	ONLY <sup>(1)</sup>	—	—	—	—
7:0	R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
	TXEN	TXABAT <sup>(2)</sup>	TXLARB <sup>(3)</sup>	TXERR <sup>(3)</sup>	TXREQ	RTREN	TXPR<1:0>	

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
-n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-21 **Unimplemented:** Read as '0'

bit 20-16 **FSIZE<4:0>:** FIFO Size bits<sup>(1)</sup>

11111 = FIFO is 32 messages deep

•  
•  
•

00010 = FIFO is 3 messages deep

00001 = FIFO is 2 messages deep

00000 = FIFO is 1 message deep

bit 15 **Unimplemented:** Read as '0'

bit 14 **FRESET:** FIFO Reset bits

1 = FIFO will be reset when bit is set, cleared by hardware when FIFO is reset. After setting, the user should poll if this bit is clear before taking any action

0 = No effect

bit 13 **UINC:** Increment Head/Tail bit

TXEN = 1: (FIFO configured as a Transmit FIFO)

When this bit is set the FIFO head will increment by a single message

TXEN = 0: (FIFO configured as a Receive FIFO)

When this bit is set the FIFO tail will increment by a single message

bit 12 **ONLY:** Store Message Data Only bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit FIFO)

This bit is not used and has no effect.

TXEN = 0: (FIFO configured as a Receive FIFO)

1 = Only data bytes will be stored in the FIFO

0 = Full message is stored, including identifier

bit 11-8 **Unimplemented:** Read as '0'

bit 7 **TXEN:** TX/RX Buffer Selection bit

1 = FIFO is a Transmit FIFO

0 = FIFO is a Receive FIFO

**Note 1:** These bits can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> bits (CiCON<23:21>) = 100).

**2:** This bit is updated when a message completes (or aborts) or when the FIFO is reset.

**3:** This bit is reset on any read of this register or when the FIFO is reset.

## Section 34. Controller Area Network (CAN)

### Register 34-20: CiFIFOCONn: CAN FIFO Control Register (n = 0 through 31) (Continued)

- bit 6     **TXABAT:** Message Aborted bit<sup>(2)</sup>  
1 = Message was aborted  
0 = Message completed successfully
- bit 5     **TXLABR:** Message Lost Arbitration bit<sup>(3)</sup>  
1 = Message lost arbitration while being sent  
0 = Message did not lose arbitration while being sent
- bit 4     **TXERR:** Error Detected During Transmission bit<sup>(3)</sup>  
1 = A bus error occurred while the message was being sent  
0 = A bus error did not occur while the message was being sent
- bit 3     **TXREQ:** Message Send Request  
TXEN = 1: (FIFO configured as a Transmit FIFO)  
Setting this bit to '1' requests sending a message.  
The bit will automatically clear when all the messages queued in the FIFO are successfully sent  
Clearing the bit to '0' while set ('1') will request a message abort.  
TXEN = 0: (FIFO configured as a Receive FIFO)  
This bit has no effect.
- bit 2     **RTREN:** Auto RTR Enable bit  
1 = When a remote transmit is received, TXREQ will be set  
0 = When a remote transmit is received, TXREQ will be unaffected
- bit 1-0   **TXPR<1:0>:** Message Transmit Priority bits  
11 = Highest Message Priority  
10 = High Intermediate Message Priority  
01 = Low Intermediate Message Priority  
00 = Lowest Message Priority

- Note 1:** These bits can only be modified when the CAN module is in Configuration mode (OPMOD<2:0> bits (CiCON<23:21>) = 100).
- 2:** This bit is updated when a message completes (or aborts) or when the FIFO is reset.
- 3:** This bit is reset on any read of this register or when the FIFO is reset.

# PIC32 Family Reference Manual

**Register 34-21: CiFIFOINTn: CAN FIFO Interrupt Register (n = 0 through 31)**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
	—	—	—	—	—	TXNFULLIE	TXHALFIE	TXEMPTYIE
23:16	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
	—	—	—	—	RXOVFLIE	RXFULLIE	RXHALFIE	RXNEMPTYIE
15:8	U-0	U-0	U-0	U-0	U-0	R-0	R-0	R-0
	—	—	—	—	—	TXNFULLIF <sup>(1)</sup>	TXHALFIF	TXEMPTYIF <sup>(1)</sup>
7:0	U-0	U-0	U-0	U-0	R/W-0	R-0	R-0	R-0
	—	—	—	—	RXOVFLIF	RXFULLIF <sup>(1)</sup>	RXHALFIF <sup>(1)</sup>	RXNEMPTYIF <sup>(1)</sup>

**Legend:**

R = Readable bit    W = Writable bit    U = Unimplemented bit, read as '0'  
 -n = Value at POR    '1' = Bit is set    '0' = Bit is cleared    x = Bit is unknown

bit 31-27 **Unimplemented:** Read as '0'

bit 26 **TXNFULLIE:** Transmit FIFO Not Full Interrupt Enable bit

- 1 = Interrupt enabled for FIFO not full
- 0 = Interrupt disabled for FIFO not full

bit 25 **TXHALFIE:** Transmit FIFO Half Full Interrupt Enable bit

- 1 = Interrupt enabled for FIFO half full
- 0 = Interrupt disabled for FIFO half full

bit 24 **TXEMPTYIE:** Transmit FIFO Empty Interrupt Enable bit

- 1 = Interrupt enabled for FIFO empty
- 0 = Interrupt disabled for FIFO empty

bit 23-20 **Unimplemented:** Read as '0'

bit 19 **RXOVFLIE:** Overflow Interrupt Enable bit

- 1 = Interrupt enabled for overflow event
- 0 = Interrupt disabled for overflow event

bit 18 **RXFULLIE:** Full Interrupt Enable bit

- 1 = Interrupt enabled for FIFO full
- 0 = Interrupt disabled for FIFO full

bit 17 **RXHALFIE:** FIFO Half Full Interrupt Enable bit

- 1 = Interrupt enabled for FIFO half full
- 0 = Interrupt disabled for FIFO half full

bit 16 **RXNEMPTYIE:** Empty Interrupt Enable bit

- 1 = Interrupt enabled for FIFO not empty
- 0 = Interrupt disabled for FIFO not empty

bit 15-11 **Unimplemented:** Read as '0'

bit 10 **TXNFULLIF:** Transmit FIFO Not Full Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

- 1 = FIFO is not full
- 0 = FIFO is full

TXEN = 0: (FIFO configured as a Receive Buffer)

Unused, reads '0'

**Note 1:** This bit is read-only and reflects the status of the FIFO.

## Section 34. Controller Area Network (CAN)

### Register 34-21: CiFIFOINTn: CAN FIFO Interrupt Register (n = 0 through 31) (Continued)

bit 9 **TXHALFIF**: FIFO Transmit FIFO Half Empty Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

1 = FIFO is  $\leq$  half full

0 = FIFO is  $>$  half full

TXEN = 0: (FIFO configured as a Receive Buffer)

Unused, reads '0'

bit 8 **TXEMPTYIF**: Transmit FIFO Empty Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

1 = FIFO is empty

0 = FIFO is not empty, at least 1 message queued to be transmitted

TXEN = 0: (FIFO configured as a Receive Buffer)

Unused, reads '0'

bit 7-4 **Unimplemented**: Read as '0'

bit 3 **RXOVFLIF**: Receive FIFO Overflow Interrupt Flag bit

TXEN = 1: (FIFO configured as a Transmit Buffer)

Unused, reads '0'

TXEN = 0: (FIFO configured as a Receive Buffer)

1 = Overflow event has occurred

0 = No overflow event occurred

bit 2 **RXFULLIF**: Receive FIFO Full Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

Unused, reads '0'

TXEN = 0: (FIFO configured as a Receive Buffer)

1 = FIFO is full

0 = FIFO is not full

bit 1 **RXHALFIF**: Receive FIFO Half Full Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

Unused, reads '0'

TXEN = 0: (FIFO configured as a Receive Buffer)

1 = FIFO is  $\geq$  half full

0 = FIFO is  $<$  half full

bit 0 **RXEMPTYIF**: Receive Buffer Not Empty Interrupt Flag bit<sup>(1)</sup>

TXEN = 1: (FIFO configured as a Transmit Buffer)

Unused, reads '0'

TXEN = 0: (FIFO configured as a Receive Buffer)

1 = FIFO is not empty, has at least 1 message

0 = FIFO is empty

**Note 1:** This bit is read-only and reflects the status of the FIFO.

# PIC32 Family Reference Manual

**Register 34-22: CiFIFOUn: CAN FIFO User Address Register (n = 0 through 31)<sup>(1)</sup>**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
CiFIFOUn<31:24>								
23:16	R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
CiFIFOUn<23:16>								
15:8	R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
CiFIFOUn<15:8>								
7:0	R-x	R-x	R-x	R-x	R-x	R-x	R-0 <sup>(2)</sup>	R-0 <sup>(2)</sup>
CiFIFOUn<7:0>								

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-0 **CiFIFOUn<31:0>**: CAN FIFO User Address bits

TXEN = 1: (FIFO configured as a Transmit Buffer)

A read of this register will return the address where the next message is to be written (FIFO head).

TXEN = 0: (FIFO configured as a Receive Buffer)

A read of this register will return the address where the next message is to be read (FIFO tail).

**Note 1:** This register is not guaranteed to read correctly in Configuration mode, and should only be accessed when the module is not in Configuration mode.

**2:** This bit will always read '0', which forces byte-alignment of messages.

**Register 34-23: CiFIFOIn: CAN Module Message Index Register (n = 0 through 31)**

Bit Range	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
31:24	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—								
23:16	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—								
15:8	U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—								
7:0	U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
CiFIFOIn<4:0>								

**Legend:**

R = Readable bit      W = Writable bit      U = Unimplemented bit, read as '0'  
 -n = Value at POR      '1' = Bit is set      '0' = Bit is cleared      x = Bit is unknown

bit 31-5 **Unimplemented:** Read as '0'

bit 4-0 **CiFIFOIn<4:0>**: CAN Side FIFO Message Index bits

TXEN = 1: (FIFO configured as a Transmit Buffer)

A read of this register will return an index to the message that the FIFO will next attempt to transmit.

TXEN = 0: (FIFO configured as a Receive Buffer)

A read of this register will return an index to the message that the FIFO will use to save the next message.

### 34.4 ENABLING AND DISABLING THE CAN MODULE

When the CAN module is OFF, the entire CAN module is held in reset, with only the CAN module Special Function Registers (SFRs) being user-accessible. The CAN module can be enabled by setting the ON bit in the CAN Control register (CiCON<15>). When the ON bit (CiCON<15>) is set, the CAN module is active and the CAN module will request priority on the CAN TX (CiTX) and RX (CiRX) pins.

Turning the CAN module OFF will place the CAN module into reset and release device control of the CiTX and CiRX pins. All message FIFOs are reset when the CAN module is turned OFF.

It may take a number of clocks for the CAN module to fully turn OFF. The CAN Module is Busy (CANBUSY) bit in the CAN Module Control register (CiCON<11>) gives a status of the CAN module. The user should poll the CANBUSY bit (CiCON<11>) to ensure that the CAN module has been turned OFF. In addition, it is important to ensure that the CAN module is in Configuration mode (see 34.5 “CAN Module Operating Modes”) before turning the CAN module OFF. This prevents bus errors caused by the CAN module being turned OFF during the transmitting of message. Example 34-1 demonstrates the necessary steps to switch the CAN1 module OFF.

#### Example 34-1: Disabling the CAN1 Module

```
/* Place the CAN module in configuration mode. */  
  
CiCONbits.REQOP = 4;  
while(CiCONbits.OPMOD != 4);  
  
/* Switch the CAN module off. */  
CiCONCLR = 0x00008000; /*Clear the ON bit */  
while(CiCONbits.CANBUSY == 1);
```

### 34.5 CAN MODULE OPERATING MODES

The CAN module can operate in one of the following modes selected by the user application:

- Configuration mode
- Normal Operation mode
- Listen-Only mode
- Listen All Messages mode
- Loopback mode
- Disable mode

The operating modes are requested by the user application that is writing to the Request Operation Mode bits (REQOP<2:0>) in the CAN Control register (CiCON<26:24>). The CAN module acknowledges entry into the requested mode by the Operation Mode bits (OPMOD<2:0>) in the CAN Control register (CiCON<23:21>). Mode transition is performed in synchronization with the CAN network.

The user application can choose to be interrupted when a requested mode change has occurred by enabling the Mode Change Interrupt (MODIE) bit in the CAN Interrupt register (CiINT<19>). When the new mode has been successfully applied, a CAN interrupt will be generated. Alternatively, the user can elect to poll the OPMOD<2:0> bits (CiCON<23:21>) to determine when the CAN module has successfully switched modes (current operation mode matches the requested operation mode).

## 34.5.1 Configuration Mode

After a device reset, the CAN module is in Configuration mode (OPMOD<2:0> bits (CiCON<23:21>) = 100). The error counters are cleared and all registers contain the reset values. The CAN module can be configured or initialized only in Configuration mode. Configuration mode is requested by programming the REQOP<2:0> bits (CiCON<26:24>) to '100'. The user application should wait till the CAN module is actually in Configuration mode. This is done by polling the OPMOD<2:0> bits (CiCON<23:21>) for a value of '100'. The following registers and bits can be modified in Configuration mode only:

- CAN Configuration register (CiCFG)
- CAN FIFO Base Address register (CiFIFOA)
- CAN Acceptance Filter Mask register (CiRXMn)
- FIFO Size (FSIZE<4:0>) bits in the CAN FIFO Control register (CiFIFOCONn<20:16>) and the Store Message Data Only (ONLY) bit (CiFIFOCONn<12>)

This protects the user from accidentally violating the CAN protocol through programming errors.

## 34.5.2 Normal Operation Mode

In Normal Operation mode, the CAN module will be on the CAN bus, and can transmit and receive CAN messages. Normal Operation mode is requested after initialization by programming the REQOP<2:0> bits (CiCON<26:24>) to '000'. When OPMOD<2:0> = 000, the CAN module proceeds with normal operation. The CAN module will check for bus idle condition before entering the Normal Operation mode.

<b>Note:</b> If the CAN module is not connected to the bus or a transceiver, the CAN module will not enter Normal Operation mode. This is because, the CAN module will always detect a dominant state at its receive pin.
---

## 34.5.3 Listen-Only Mode

The Listen-only mode is a variant of Normal Operation mode. If Listen-Only mode is activated, the CAN module is present on the CAN bus but is passive. It will receive messages but not transmit. The CAN module will not generate error flags and will not acknowledge signals. The error counters are deactivated in this state. Listen-Only mode can be used for detecting the baud rate on the CAN bus. For this to occur, it is necessary that at least two other nodes are present that communicate with each other. The baud rate can be detected empirically by testing different values. This mode is also useful as a bus monitor, as the CAN bus does not influence the data traffic.

## 34.5.4 Listen All Messages Mode

The Listen All Messages mode is a variant of Normal Operation mode. If Listen All Messages mode is activated, transmission and reception operate the same as normal operation mode with the exception that if a message is received with an error, it will be transferred to the receive buffer as if there was no error. The receive buffer will contain data that was received up to the error. The filters still need to be enabled and configured.

## 34.5.5 Loopback Mode

Loopback mode is used for self-test to allow the CAN module to receive its own message. In this mode, the CAN module transmit path is connected internally to the receive path. A "dummy" acknowledge is provided, thereby eliminating the need for another node to provide the Acknowledge bit. The CAN message is not actually transmitted on the CAN bus.

## 34.5.6 Disable Mode

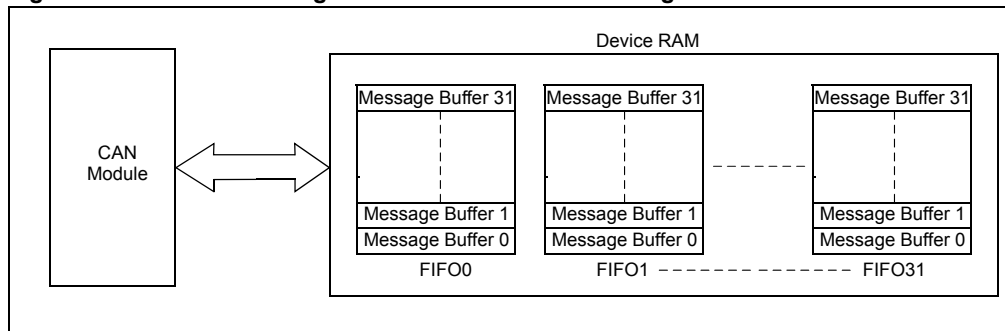
Disable mode is similar to Configuration mode, except that the CAN module error counters are not reset. The CAN module is placed in Disable mode by setting the REQOP<2:0> bits (CiCON<26:24>) to '001'. In Disable mode, the CAN module's internal clock will stop unless the CAN module is receiving or transmitting a message. The CAN module will not be allowed to enter Disable mode while a transmission or reception is taking place to prevent the CAN module from causing errors on the system bus. The CAN module will enter Disable mode when the current message completes. The OPMOD<2:0> bits (CiCON<23:21>) indicate whether the CAN module successfully went into Disable mode. The CAN module Transmit (CiTX) pin will stay in the recessive state while the CAN module is in Disable mode to prevent inadvertent CAN bus errors.



## 34.6 CAN MESSAGE HANDLING

The CAN module uses device RAM for storing CAN messages that need to be transmitted or received. The CAN module by itself does not have user-accessible message buffers. [Figure 34-8](#) illustrates the organization of the CAN module buffer memory in device RAM.

**Figure 34-8: CAN Message Buffers and Device RAM Organization**



As illustrated in [Figure 34-8](#), the CAN module organizes message buffers as FIFOs. A total of 32 distinct FIFOs are available, which have the following characteristics:

- Minimum size of one CAN message buffer and a maximum size of 32 CAN message buffers
- Independent configurable size
- Configurable to be a transmit message or a receive message FIFO
- User-readable head and tail pointer
- Independently configurable interrupts
- Status bits to provide the status of the FIFO as messages are transmitted or received
- Can be a transmit FIFO or receive FIFO, but not both (therefore, if a FIFO is configured for transmit operation, all messages in the FIFO are considered for transmission)
- Can be configured to be a transmit FIFO or receive FIFO, independent of each other and can be configured in any order

Each of the 32 message FIFOs has the following associated registers:

- CAN FIFO Control register (CiFIFOCONn) ([Register 34-20](#))
- CAN FIFO Interrupt register (CiFIFOINTn) ([Register 34-21](#))
- CAN FIFO User Address register (CiFIFOUAN) ([Register 34-22](#))
- CAN FIFO Message Index register (CiFIFOIn) ([Register 34-23](#))

The CAN module requires only the start address of the first message buffer in the FIFO. The CAN FIFO Base Address register (CiFIFOBA) should point to the start address of this message buffer. The CAN module automatically calculates the address of message buffers in each of the FIFO based on the configuration of the individual FIFOs. The individual FIFOs will be arranged contiguously, with all the message buffers being packed. This produces a CAN message buffer memory without any gaps. While the CiFIFOUAn register provides the address of the next read/write CAN Message Buffer that the user application must use, the CiFIFOCIn register provides the corresponding CAN Message Buffer Index of the next message buffer to be transmitted or written to by the CAN module.

For more information on FIFO related interrupts along with other CAN module interrupts, refer to [34.12 “CAN Interrupts”](#).

## 34.6.1 Message FIFO Configuration

The user application must allocate device RAM space for CAN message buffering. The requirements of each type of message buffer are as follows:

- A CAN transmit message buffer requires 4 words (16 bytes) of memory
- A CAN receive message buffer will require 4 words (16 bytes) of memory, if the entire message (time stamp plus data plus message ID) is stored - (full receive message)
- A CAN receive message buffer will require 2 words (8 bytes) of memory only if the data is stored - (data-only receive message)

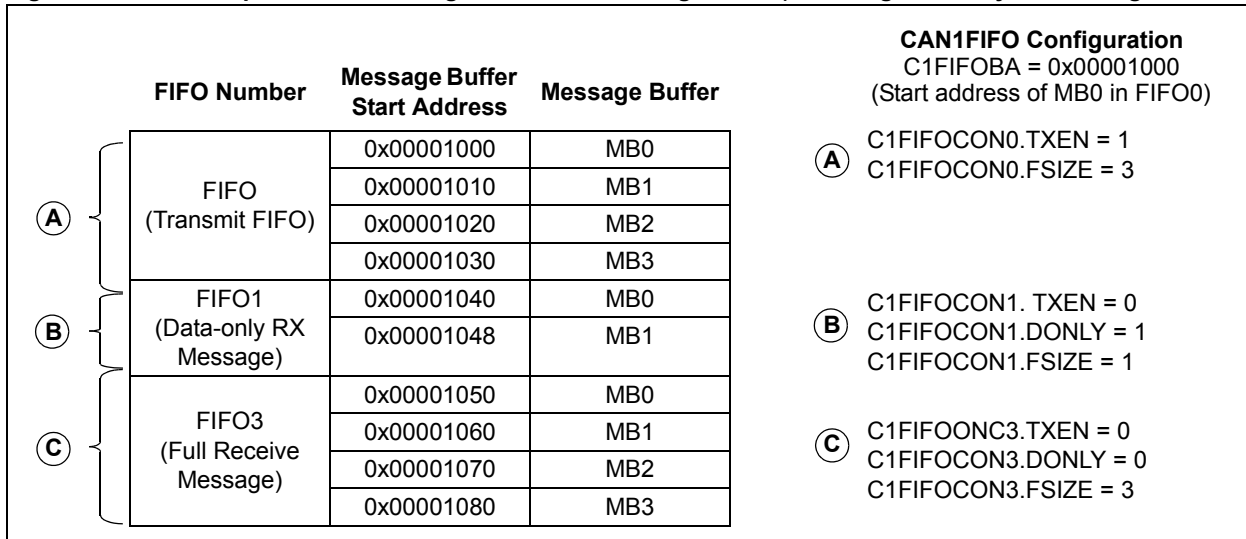
The user application must design the size of each message FIFO. The FIFO size is controlled through the FIFO size (FSIZE<4:0>) bits in the corresponding CAN FIFO Control register (CiFIFOCOnn<20:16>). All FIFOs have a default size of at least one message buffer. The total memory to be allocated is obtained by counting the total number of buffers across FIFOs while accounting for the memory size of each buffer.

For optimal use of the CAN FIFO message buffering model, the user application should configure the required number of FIFOs in the order starting from FIFO0, and then configuring FIFO1, FIFO2, FIFO3, and so on.

For example, a user application requires 29 FIFOs and configures FIFO0, FIFO1 and FIFO5 through FIFO31. FIFO2, FIFO3 and FIFO4 are not configured. FIFO2, FIFO3 and FIFO4, although not configured, still occupy 4 words of RAM each. The CAN module will not use these FIFOs, but takes these into account while computing the address of FIFO5. In this example, the user application must allocate memory for all of the 32 FIFOs. The memory occupied by FIFO2, FIFO3 and FIFO4 could be used by the user application.

An optimal way to accomplish this is to configure FIFO0 through FIFO28, and not configure FIFO29, FIFO30 and FIFO31. Therefore, the user application must allocate space only for 29 FIFOs. [Figure 34-9](#) illustrates an example.

**Figure 34-9: Example of CAN Message Buffer FIFO Configuration (Including Data-only RX Message Buffers)**



## Section 34. Controller Area Network (CAN)

For the CAN1 message buffer FIFO configuration illustrated in [Figure 34-9](#), FIFO0 has four transmit buffers, which would require a total of 16 (4 buffers x 4 words per buffer) words. FIFO1 has two data-only receive buffers, which would require a total of four (2 buffers x 2 words per buffer) words. FIFO3 has four full receive buffers, which would require a total of 16 (4 buffers x 4 words per buffer) words. Therefore, a total of 36 (16 + 4 + 16) words of memory needs to be allocated.

Consider another example of a user application illustrated in [Figure 34-10](#), which requires a total of four FIFOs.

**Figure 34-10: Example of CAN FIFO Configuration**

	FIFO Number	Message Buffer Start Address	Message Buffer	CAN1FIFO Configuration
A	FIFO0	0x00002000	MB0	C1FIFOCON0.TXEN = 1 C1FIFOCON0.SIZE = 1
		0x00002010	MB1	
B	FIFO1	0x00002020	MB0	C1FIFOCON1.TXEN = 1 C1FIFOCON1.SIZE = 1
		0x00002030	MB1	
C	FIFO2	0x00002040	MB0	C1FIFOCON2.TXEN = 0 C1FIFOCON2.SIZE = 2
		0x00002050	MB1	
		0x00002060	MB2	
D	FIFO3	0x00002070	MB0	C1FIFOCON3.TXEN = 0 C1FIFOCON3.SIZE = 2
		0x00002080	MB1	
		0x00002090	MB2	

FIFO0 and FIFO1 have two message buffers each and are configured to be CAN message transmit FIFOs. FIFO2 and FIFO3 have three message buffers each and are configured to be CAN message receive FIFOs. FIFO0 and FIFO1 will require a total of 16 (2 FIFOs x 2 message buffers per FIFO x 4 words per message buffer) words of memory. FIFO2 and FIFO3 will require a total of 24 (2 FIFOs x 3 message buffers per FIFO x 4 words per message buffer) words of memory. Therefore, a total of 40 (16 + 24) words of memory needs to be allocated.

The following steps can be used to configure the CAN module FIFOs:

1. Allocate memory for the CAN message buffer FIFOs.
2. Place the CAN module into Configuration mode (OPMOD<2:0> = 100).
3. Update the CAN FIFO Base Address register (CiFIFOBA) with the base address of the FIFO. This should be the physical start address of Message Buffer 0 of FIFO0.
4. Update the FSIZE<4:0> bits (CiFIFOCONn<20:16>).
5. Select whether the FIFO is to be a transmit or receive FIFO (TXEN bit (CiFIFOCONn<7:0>)).
6. Place the CAN module into Normal Operation mode (OPMOD<2:0> = 000).

[Example 34-2](#) illustrates how the above coding steps can be used to configure the CAN message FIFO as shown in [Example 34-10](#).

## Example 34-2: Setting Up the CAN Message FIFO

```
/* This code snippet illustrates the steps required to configure the */
/* PIC32 CAN Message FIFOs. The FIFO configuration example shown in */
/* Figure 34-5 will be used in this example. */

/* FIFO0 - Transmit - 2 MESSAGE BUFFERS */
/* FIFO1 - Transmit - 2 MESSAGE BUFFERS */
/* FIFO2 - Receive - 3 MESSAGE BUFFERS */
/* FIFO3 - Receive - 3 MESSAGE BUFFERS */
/* FIFO4 - FIFO31 - Not used */

/* Allocate a total of 40 words.
unsigned int CanFifoMessageBuffers[40];

/* Request CAN to switch to configuration mode and wait until it has switched */

C1CONbits.REQOP = 100
while(C1CONbits.OPMOD != 100);

/* Initialize C1FIFоба register with physical address of CAN message Buffer */
C1FIFоба = KVA_TO_PA(CanFifoMessageBuffers); ;

/* Configure FIFO0 */
C1FIFOCON0bits.FSIZE = 1;
C1FIFOCON0SET = 0x80;          /* Set the TXEN bit */

/* Configure FIFO1 */
C1FIFOCON1bits.FSIZE = 1;
C1FIFOCON1SET = 0x80;          /* Set the TXEN bit */

/* Configure FIFO2 */
C1FIFOCON2bits.FSIZE = 2;
C1FIFOCON2CLR = 0x80;          /* Clear the TXEN bit */

/* Configure FIFO3 */
C1FIFOCON3bits.FSIZE = 2;
C1FIFOCON3CLR = 0x80;          /* Clear the TXEN bit */

/* The CAN module can now be placed into normal mode if no further */
/* configuration is required. */

C1CONbits.REQOP = 0;

while(C1CONbits.OPMOD != 0);
```

### 34.6.2 Loading Messages to be Transmitted into the FIFO

In order to use a FIFO to transmit data, the FIFO must be configured as a transmit FIFO. This is done by setting the TX/RX Buffer Selection (TXEN) bit in the CAN FIFO Control register (CiFIFOCONn<7>).

The CAN FIFO User Address register (CiFIFOUAn) provides the physical address of the next message buffer in FIFO where the user application should store the message (also referred to as the FIFO head location). The CAN message should be loaded, starting at the location provided by the CiFIFOUAn register.

The user application should set the Increment Head/Tail (UINC) bit in the CAN FIFO Control register (CiFIFOCONn<13>) after loading one message buffer in the FIFO. This will cause the CAN module to increment the address contained in the CiFIFOUAn register by 16 (thereby pointing to the next message buffer). The message is then ready to be transmitted. The CiFIFOUAn register rolls over to the start of the FIFO when it has reached the end of FIFO.

As a result, the user application need not track the FIFO head location, and can use the CiFIFOUAn register for this purpose. The following coding steps can be followed to load a message for transmission into a transmit FIFO:

1. Read the CiFIFOUAn register and store one message (16 bytes) at this address.
2. Set the UINC bit (CiFIFOCONn<13>) to update the CiFIFOUAn register.
3. Set the Message Send Request (TXREQ) bit in the CAN FIFO Control register (CiFIFOCONn<3>) to transmit the message.
4. Perform steps 2 and 3 for the number of messages to be queued and the size of the FIFO.

**Example 34-3** illustrates the above coding steps. In this example, four messages are loaded into FIFO0 of the CAN1 module.

#### Example 34-3: Loading a Transmit Message FIFO

```
/* This code snippet illustrates the steps to load a transmit message FIFO. */
/* This example uses the CAN1 module. */

/* Four messages to be transmitted using transmit FIFO0 */

int message; /* Tracks the message buffer */
unsigned int * currentMessageBuffer; /* Points to message buffer to be written */

message = 0;

for(message = 0; message <= 3; message++)
{
    /* Get the address of the message buffer to write to from the C1FIFOUA0 */
    /* register. Convert this physical address to virtual address. */

    currentMessageBuffer = PA_TO_KVA1(C1FIFOUA0);

    /* This procedure will load the message
    * buffer with the message to be transmitted. */

    LoadMessageBuffer(currentMessageBuffer);

    C1FIFOCON0SET = 0x2008; /* Set the UINC and TXREQ bit */
}

/* At this point the messages are loaded in FIFO0 */
```

## 34.6.3 Accessing Received Messages In the FIFO

In order to use the FIFO to receive data, the FIFO must be configured as a receive FIFO. This is done by clearing the TXEN bit (CiFIFOCONn<7>). After a message is received, the user application should obtain the physical start address of the first message buffer to read (also called the FIFO tail pointer) from the CiFIFOUAn register. The message can then be read from this address onward.

After processing the message from the FIFO, the user application should signal the CAN module that the message has been processed and can be overwritten by setting the UINC bit (CiFIFOCONn<13>). This will increment the tail pointer and increase the address pointed to by the CiFIFOUAn register by 4 words or 2 words, depending on the value of the DONLY bit (CiFIFOCONn<12>). The CiFIFOUAn register rolls over to the start of the FIFO when it has reached the end of the FIFO.

As a result, the user application need not track the FIFO tail location, and can use the CiFIFOUAn register for this purpose. The following coding steps can be followed to process a message in a receive FIFO:

1. Use any of the available interrupts to determine if there is a message in the FIFO.
2. Read the CiFIFOUAn register and process one message (16 bytes) from this address.
3. Set the UINC bit (CiFIFOCONn<13>) to update the CiFIFOUAn register.
4. Perform steps 1 and 2 until the interrupt condition gets cleared.

**Example 34-4** illustrates the code using the above steps. In this code example, FIFO1 of the CAN 1 module is configured for receive operation. The code example reads the FIFO until it is empty.

### Example 34-4: Reading Received Messages from the FIFO

```
/* This code snippet illustrates the steps to read messages from receive */
/* message FIFO. This example uses the CAN1 module.*/

/* FIFO1 size is 4 messages and each message is 4 words long. */
unsigned int * currentMessageBuffer; /* Points to message buffer to be read */

while(1)
{
    /* Keep reading till the FIFO is empty. */
    while(C1FIFOINT1bits.RXEMPTYIF == 1)
    {
        /* Get the address of the message buffer to read from the C1FIFOUA1 */
        /* register. Convert this physical address to virtual address. */

        currentMessageBuffer = PA_TO_KVA1(C1FIFOUA1);

        ProcessReceivedMessage(currentMessageBuffer);

        /* Set the UINC bit to tell the CAN module that
         * a message has been read. */

        C1FIFOCON0SET = 0x2000;
    }
}
```

### 34.6.4 Resetting the FIFO

The FIFO can be reset by any of the following methods:

- Setting the FIFO Reset (FRESET) bit in the CAN FIFO Control register (CiFIFOCONn<14>)
- Placing the CAN module into Configuration mode (OPMOD<2:0> bits (CiCON<23:21>) = 100)
- Turning the CAN module OFF (CiCON<15> = 0)

Resetting the FIFO will reset the head and tail pointers, the interrupt flags, and the following status bits in the CAN FIFO Control register (CiFIFOCONn): the Message Aborted (TXABAT) bit (CiFIFOCONn<6>), the Message Lost Arbitration (TXLARB) bit (CiFIFOCONn<5>) and the Error Detected During Transmission (TXERR) bit (CiFIFOCONn<4>).

The following should be considered before resetting a FIFO using the FRESET bit (CiFIFOCONn<14>):

- If the FIFO is a transmit FIFO, no transmissions should be pending
- If the FIFO is a receive FIFO, it should not be pointed to by any active filters

A user application may typically reset the FIFO after coming out of Disable mode. While resetting the FIFO using the FRESET bit (CiFIFOCONn<14>), the user application must poll the bit to ensure that the reset operation has completed. This is shown in [Example 34-5](#).

#### Example 34-5: Resetting a Message FIFO

```
/* This code snippet shows how to reset a message FIFO. This example */
/* uses FIFO0 of the CAN1 module. */

C1FIFOCON0SET = 0x00004000;          /* Set the FRESET bit */
while(C1FIFOCON0bits.FRESET == 1);
```

## 34.7 TRANSMITTING A CAN MESSAGE

The CAN module will transmit messages that are loaded in a transmit FIFO. For detailed descriptions on configuring a message FIFO for transmit operation, refer to [34.6.1 “Message FIFO Configuration”](#) and [34.6.2 “Loading Messages to be Transmitted into the FIFO”](#).

### 34.7.1 Format of Transmit Message Buffer

The transmit message FIFO can hold up to 32 CAN transmit message buffers. A transmit message buffer is 4 words (16 bytes) long and has a fixed format as described in [Table 34-2](#). It contains four words: CMSGSID, CMSGEID, CMSGDATA0 and CMSGDATA1. The bits in these registers have a one-to-one correspondence to bit fields in the CAN message, as defined by the CAN Specification 2.0B. The user application must ensure that every message buffer in the transmit FIFO conforms to the formats illustrated in [Figure 34-11](#) through [Figure 34-14](#).

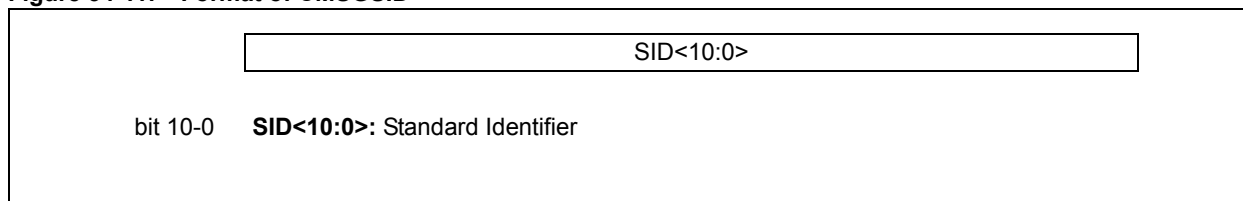
**Table 34-2: Transmit Message Buffer Format as Stored in System Memory**

Address Offset	Name	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0		
00	CMSGSID	31:24	---	---	---	---	---	---	---		
		23:16	---	---	---	---	---	---	---		
		15:8	---	---	---	---	---	SID<10:8>			
		7:0	SID<7:0>								
04	CMSGEID	31:24	---	---	SRR	IDE	EID<17:14>				
		23:16	EID<13:6>								
		15:8	EID<5:0>						RTR	RB1	
		7:0	---	---	---	RB0	DLC<3:0>				
08	CMSGDATA0	31:24	Transmit Buffer Data Byte 3								
		23:16	Transmit Buffer Data Byte 2								
		15:8	Transmit Buffer Data Byte 1								
		7:0	Transmit Buffer Data Byte 0								
0C	CMSGDATA1	31:24	Transmit Buffer Data Byte 7								
		23:16	Transmit Buffer Data Byte 6								
		15:8	Transmit Buffer Data Byte 5								
		7:0	Transmit Buffer Data Byte 4								

**Legend:** Shaded bits should be set to '0'.

**Note 1:** The CAN transmit message is stored in device RAM and does not have SET/CLR/INV registers associated with it.

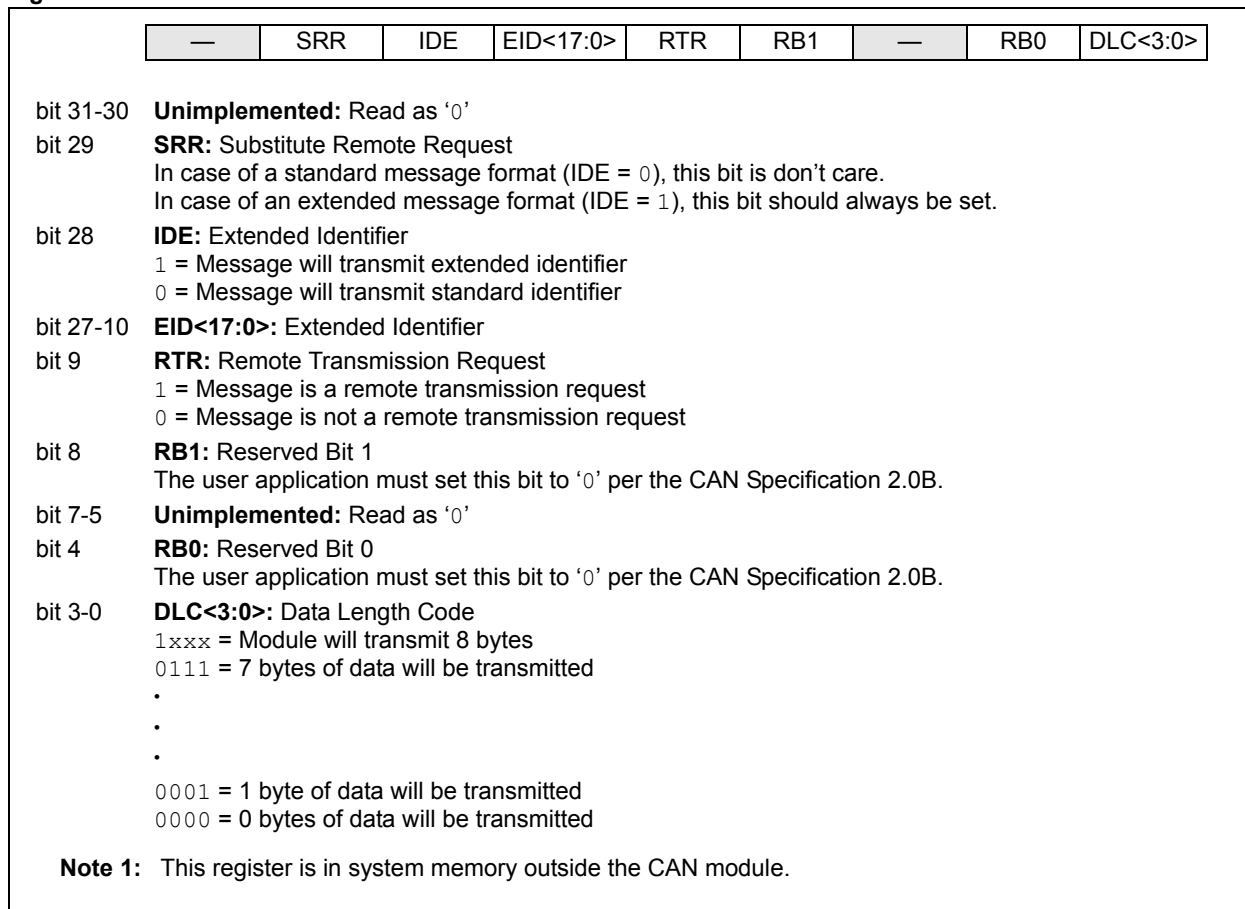
**Figure 34-11: Format of CMSGSID**



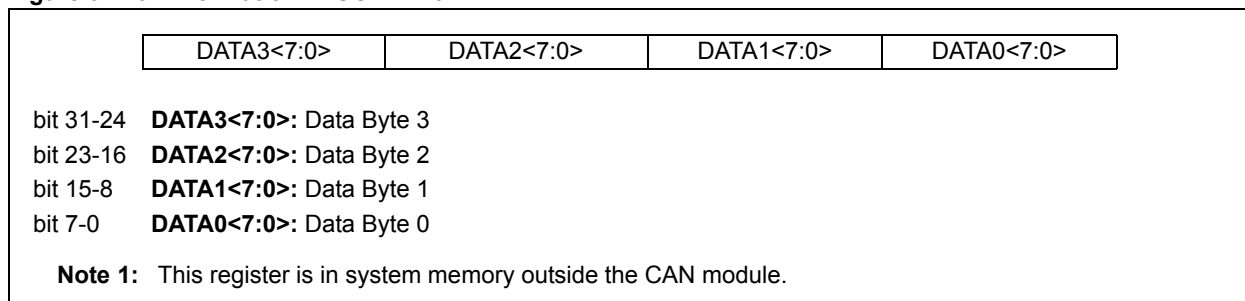


## Section 34. Controller Area Network (CAN)

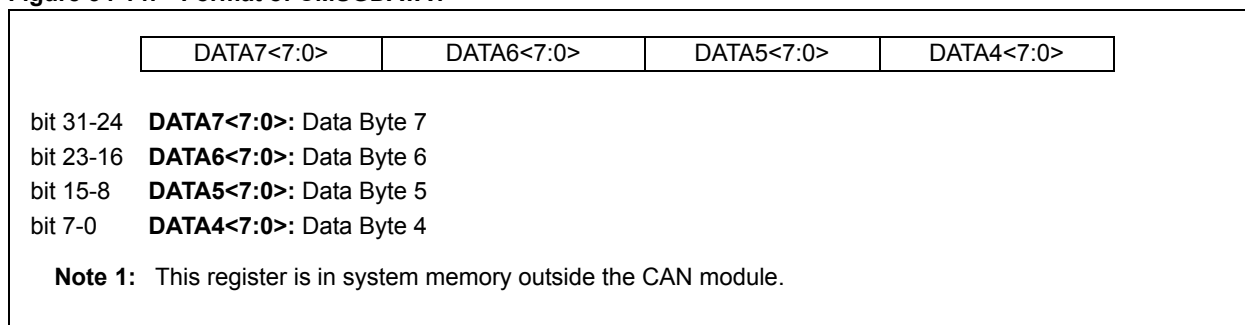
**Figure 34-12: Format of CMSGEID**



**Figure 34-13: Format of CMSGDATA0**



**Figure 34-14: Format of CMSGDATA1**



[Example 34-6](#) shows a code example data structure for implementing a CAN message buffer in memory. An example of using this structure is provided in [Example 34-7](#).

## Example 34-6: Implementing a CAN Message Buffer in Memory

```
/* This code snippet shows an example of data structure to implement a CAN */
/* message buffer. */

/* Define the sub-components of the data structure as specified in Table 34-2 */

/* Create a CMSGSID data type. */
typedef struct
{
    unsigned SID:11;
    unsigned :21;
}txcmsgsid;

/* Create a CMSGEID data type. */
typedef struct
{
    unsigned DLC:4;
    unsigned RB0:1;
    unsigned :3;
    unsigned RB1:1;
    unsigned RTR:1;
    unsigned EID:18;
    unsigned IDE:1;
    unsigned SRR:1;
    unsigned :2;
}txcmsgeid;

/* Create a CMSGDATA0 data type. */
typedef struct
{
    unsigned Byte0:8;
    unsigned Byte1:8;
    unsigned Byte2:8;
    unsigned Byte3:8;
}txcmsgdata0;

/* Create a CMSGDATA1 data type. */
typedef struct
{
    unsigned Byte4:8;
    unsigned Byte5:8;
    unsigned Byte6:8;
    unsigned Byte7:8;
}txcmsgdatal;

/* This is the main data structure. */
typedef union uCANTxMessageBuffer {

    struct
    {
        txcmsgsid CMSGSID;
        txcmsgeid CMSGEID;
        txcmsgdata0 CMSGDATA0;
        txcmsgdatal CMSGDATAL;
    };
    int messageWord[4];
}CANTxMessageBuffer;
```

## Section 34. Controller Area Network (CAN)

### Example 34-7: Example Usage of the Data Structure

```
/* Example usage of data structure shown in Example 34-6. This example sets */
/* up a message buffer in FIFO1 */

CANTxMessageBuffer * buffer;

buffer = (CANTxMessageBuffer *) (PA_TO_KVA1(CiFIFOU1));

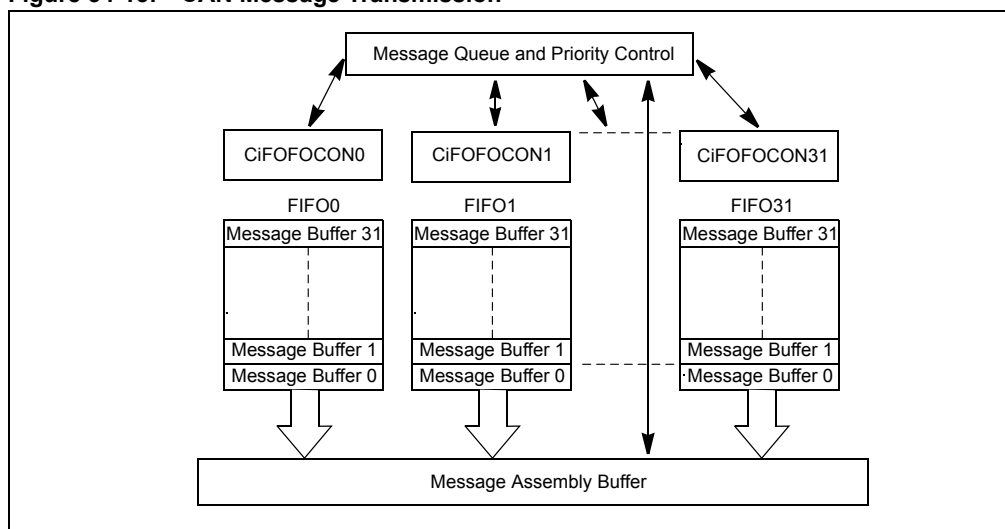
/* Clear the message buffer. */
buffer->messageWord[0] = 0;
buffer->messageWord[1] = 0;
buffer->messageWord[2] = 0;
buffer->messageWord[3] = 0;

buffer->CMSGSID.SID = 0x100; /* Message SID */
buffer->CMSGEID.DLC = 0x2; /* Data Length is 2 */
buffer->CMSGDATA0.Byte0 = 0xAA; /* Byte 0 Data */
buffer->CMSGDATA0.Byte1 = 0xbb; /* Byte 1 Data */
```

### 34.7.2 Requesting Transmit of a Message

Table 34-15 shows the CAN module transmission process. The user application must configure the FIFO for transmit operation. Message transmission and priority is controlled by the CiFIFOCONn register (see Register 34-20). The CAN module will then select the next message to be transmitted based on readiness and priority, and will process this message for transmission.

Figure 34-15: CAN Message Transmission



Once a message has been loaded into the FIFO and is ready to be transmitted, the user application can initiate a transmission by setting the TXREQ bit (CiFIFOCONn<3>). When this bit is set, the contents of the FIFO will be queued for transmission according to the message priority, and the CAN module will transmit all the messages in the FIFO. When all messages have been transmitted, the TXREQ bit (CiFIFOCONn<3>) will be cleared. A user application can load all of the transmit FIFOs and set the TXREQ bit (CiFIFOCONn<3>) for all of these FIFOs. Alternately, the user application can load one transmit FIFO, set the associated TXREQ bit (CiFIFOCONn<3>), and then load the next FIFO while the previous FIFO is being processed.

Messages can be appended to the FIFO while a message is being transmitted. The user application should use the CiFIFOUn register to get the FIFO head pointer and should store the message at this address. Appended messages will be queued for transmission.

**Note:** It is recommended that the user application set the TXREQ bit (CiFIFOCONn<3>) after every message is loaded into the FIFO (after the UINC bit (CiFIFOCONn<13>) is set).

The user application should check that the transmit FIFO can accommodate messages (that is, the transmit FIFO is not full) before writing messages to the FIFO. This can be done by checking the TxNFULLIF bit in the CAN FIFO Interrupt register (CiFIFOINTn<10>). Writing a message to a FIFO which is full could cause an untransmitted message to be overwritten.

The following coding steps can be used to transmit CAN messages:

1. Configure the desired number of FIFOs for transmit operation.
2. If desired, set the priority of the individual FIFOs.
3. Create a transmit message using the format shown in [Table 34-2](#).
4. If the TxNFULLIF bit (CiFIFOINTn <10>) is clear, this means the FIFO is full. In this case, skip to Step 7.
5. Read the CiFIFOUn register and store the message at the provided address.
6. Set the UINC bit (CiFIFOCONn<13>).
7. Set the TXREQ bit (CiFIFOCONn<3>).
8. Repeat steps 3 through 6 for the number of messages to be transmitted across the desired number of FIFOs.
9. The transmit FIFO interrupts can be used to monitor the FIFO status.

[Example 34-8](#) shows a code example that provides the steps required to transmit a CAN message using the CAN1 module to transmit four messages. Message 0 and 1 are Standard Identifier (SID) CAN messages. Message 2 and 3 are EID messages. FIFO1 is used and its length is 3.

#### Example 34-8: Transmitting Standard and Extended ID Messages

```
/* This code example illustrates how to transmit standard and extended */
/* ID messages with the PIC32 CAN module. */

/* The code example uses CAN1, FIFO0. FIFO0 size is 4 */

/* Four CAN message have to be transmitted. */
/* Msg 0: SID - 0x100 Data - 0x12BC1245 */
/* Msg 1: SID - 0x102 Data - 0x12BC124512BC1245 */
/* Msg 2: SID - 0x100 EID - 0xC000 Data - 0x12BC1245 */
/* Msg 3: SID - 0x102 EID - 0xC000 Data - 0x12BC124512BC1245 */

/* Pointer to CAN Message Buffer */
CANTxMessageBuffer * transmitMessage;

/* This array is the CAN FIFO and message buffers. FIFO has 4 message */
/* buffers. All other buffers are default size. */

unsigned int CANFIFO[16];

/* Place CAN module in configuration mode */

C1CONbits.REQOP = 4;
while(C1CONbits.OPMOD != 4);

/* Initialize the C1FIFOBA with the start address of the CAN FIFO message */
/* buffer area. */

C1FIFOBA = KVA_TO_PA(CANFIFO);

/* Set FIFO0 size to 3 messages. All other FIFOs at default size. */
/* Configure FIFO0 for transmit.*/

C1FIFOCON0bits.FSIZE = 2
C1FIFOCON0SET = 0x00000080; /* Set the TXEN bit - Transmit FIFO */
```

## Section 34. Controller Area Network (CAN)

### Example 34-8: Transmitting Standard and Extended ID Messages (Continued)

```
/* Place the CAN module in Normal mode. */

ClCONbits.REQOP = 0;
while(ClCONbits.OPMOD != 0);

/* Get the address of the message buffer to write to. Load the buffer and */
/* then set the UINC bit. Set the TXREQ bit next to send the message. */

/* Message 0 SID - 0x100 Data - 0x12BC1245*/

transmitMessage = (CANTxMessageBuffer *) (PA_TO_KVA1(ClFIFOA1));
transmitMessage->CMSGSID.SID = 0x100;          /* CMSGSID */
transmitMessage->CMSGEID.IDE = 0;
transmitMessage->CMSGEID.DLC = 0x4;
transmitMessage->messageWord[2] = 0x12BC1245;  /* CMSGDAT0 */
ClFIFOCON1SET = 0x00002000;                    /* Set the UINC bit */
ClFIFOCON1SET = 0x00000008;                    /* Set the TXREQ bit */

/* Message 1 SID - 0x102 Data - 0x12BC124512BC1245 */

transmitMessage = (CANTxMessageBuffer *) (PA_TO_KVA1(ClFIFOA1));
transmitMessage->CMSGSID.SID = 0x102;          /* CMSGSID */
transmitMessage->CMSGEID.IDE = 0;
transmitMessage->CMSGEID.DLC = 0x8;
transmitMessage->messageWord[2] = 0x12BC1245;  /* CMSGDAT0 */
transmitMessage->messageWord[3] = 0x12BC1245;  /* CMSGDAT1 */
ClFIFOCON1SET = 0x00002000;                    /* Set the UINC bit */
ClFIFOCON1SET = 0x00000008;                    /* Set the TXREQ bit */

/* Message 2 SID - 0x100 EID - 0xC000 Data - 0x12BC1245*/

transmitMessage = (CANTxMessageBuffer *) (PA_TO_KVA1(ClFIFOA1));
transmitMessage->CMSGSID.SID = 0x100;          /* CMSGSID */
transmitMessage->CMSGEID.SID = 0xC000;         /* CMSGEID */
transmitMessage->CMSGEID.IDE = 1;
transmitMessage->CMSGEID.DLC = 0x4;
transmitMessage->messageWord[2] = 0x12BC1245;  /* CMSGDAT0 */
ClFIFOCON1SET = 0x00002000;                    /* Set the UINC bit */
ClFIFOCON1SET = 0x00000008;                    /* Set the TXREQ bit */

/* Message 3 SID - 0x102 EID - 0xC000 Data - 0x12BC124512BC1245*/

transmitMessage = (CANTxMessageBuffer *) (PA_TO_KVA1(ClFIFOA1));
transmitMessage->CMSGSID.SID = 0x102;          /* CMSGSID */
transmitMessage->CMSGEID.SID = 0xC000;         /* CMSGEID */
transmitMessage->CMSGEID.IDE = 1;
transmitMessage->CMSGEID.DLC = 0x8;
transmitMessage->messageWord[2] = 0x12BC1245;  /* CMSGDAT0 */
transmitMessage->messageWord[3] = 0x12BC1245;  /* CMSGDAT1 */
ClFIFOCON1SET = 0x00002000;                    /* Set the UINC bit */
ClFIFOCON1SET = 0x00000008;                    /* Set the TXREQ bit */

}
```

## 34.7.3 Transmit Message Priority

The CAN module allows the user application to control the priority of transmit message buffers. Prior to sending a message SOF, the CAN module compares the priority of all buffers that are ready for transmission. The transmit message buffer with the highest priority will be sent first. For example, if transmit FIFO0 has a higher priority setting than transmit FIFO1, all messages in FIFO0 will be sent first. In a case where the priority of a FIFO is changed while another FIFO is being processed, the CAN module will reassess the priority of all FIFOs after it has completed transmitting the current message. This allows the change in FIFO priority to take effect at the earliest opportunity.

The priority levels are controlled by the Message Transmit Priority (TXPR<1:0>) bits in the CAN FIFO Control register (CiFIFOCONn<1:0>). There are four levels of transmit priority as defined by the (TXPR<1:0>) bits (CiFIFOCONn<1:0>). The selections are as follows:

- 11 = This FIFO has the highest priority
- 10 = This FIFO has an intermediate high priority
- 01 = This FIFO has an intermediate low priority
- 00 = This FIFO has the lowest priority

If two FIFOs have the same priority setting, the FIFO with the highest natural order is sent. The natural order of transmission if all FIFOs have the same priority is as follows:

- FIFO0 = lowest natural priority
- FIFO1 = higher natural priority
- 
- 
- 
- FIFO31 = highest natural priority

## 34.7.4 Aborting Transmission of a Queued Message

A message that has been queued to be transmitted can be aborted by clearing the TXREQ bit (CiFIFOCONn<3>). If the TXREQ bit (CiFIFOCONn<3>) is cleared, the CAN module will attempt to abort the transmission.

If the message aborts successfully, the TXABAT bit (CiFIFOCONn<6>) will be set by hardware. TXREQ will remain set until the message either aborts or is successfully transmitted.

If the message is successfully transmitted, the FIFO pointers will be updated as normal. If the message is successfully aborted, the FIFO pointers will not change. The user can then use the internal index (CFIFOCI) to determine which messages have already been transmitted if required.

To reset the FIFO pointers and erase all pending messages, the user application can set the FRESET bit (CiFIFOCONn<14>). The FIFO can then be re-enabled and loaded with new messages to be transmitted.

### 34.7.5 Remote Transmit Request

As discussed in [34.7.2 “Requesting Transmit of a Message”](#), the CAN bus system has a method for allowing a node to request data from another node. The requesting node sends a message with the RTR bit set. The message contains no data, only an address to trigger a filter match.

The filter that is configured to respond to an RTR will point to a FIFO that is configured for transmission. The FIFO must also be enabled to reply to the remote transmission requests by setting the RTREN = 1.

RTRs can be handled without CPU intervention. If a transmit FIFO is configured correctly, when a filter matches and that filter points to the FIFO, the buffer will be queued for transmission. The FIFO must be configured as follows:

1. Set FIFO to Transmit mode by setting the TXEN bit (CiFIFOCONn<7>) to ‘1’.
2. A filter must be enabled and loaded with a matching message identifier.
3. The buffer pointer register for that filter must point to the transmit buffer (note that although a filter normally points to a receive FIFO, in this case it must point to the transmit FIFO).
4. The RTREN bit (CiFIFOCONn<2>) must be set to ‘1’ to enable RTR.
5. The FIFO must be preloaded with at least one message to be sent.

When a RTR message is received, and it matches a filter pointing to the properly configured transmit buffer, the TXREQ bit (CiFIFOCONn<3>) is automatically set. The message buffers in the FIFO are then transmitted in priority order. If no messages are available in the transmit buffer when a request for a remote transmission occurs, the event will be treated as a FIFO overflow, and the Receive FIFO Overflow Interrupt Flag (RXOVFLIF) bit in the CAN FIFO Interrupt register (CiFIFOINTn<3>) will be set. [Example 34-9](#) shows a code example of how a node can request data from remote node. [Example 34-10](#) shows a code example of how a node can be configured to respond to a RTR.

#### Example 34-9: Remote Transmit Request

```
/* This code snippet shows an example of a Remote Transmit Request. The */
/* node in this case will request a transmission from an application (or */
/* node) with an SID of 0x100. */

CANMessageBuffer * rtrMessage;
rtrMessage = (CANMessageBuffer *) (PA_TO_KVA1(C1FIFOA1));

/* Clear the message. */
rtrMessage->messageWord[0] = 0x0;
rtrMessage->messageWord[1] = 0x0;
rtrMessage->messageWord[2] = 0x0;
rtrMessage->messageWord[3] = 0x0;

rtrMessage->CMSGSID.SID = 0x100;
rtrMessage->CMSGEID.IDE = 0;
rtrMessage->CMSGEID.RTR = 1;
rtrMessage->CMSGEID.DLC = 0;
C1FIFOCON1SET = 0x00002000; /* Set the UINC bit */
/* The Remote Transmit Request message (rtrMessage) is now ready to be sent.*/
```

## Example 34-10: Responding to a Remote Transmit Request

```
/* This code example shows how to configure the CAN1 module to respond */
/* to a remote transmit request. */

/* In this case, FIFO1 is configured to respond to the a remote request */
/* on SID = 0x100. */

/* Allocate CAN FIFO memory. */
unsigned int CANFIFO[140];

/* This is the pointer to the reply message. */
CANMessageBuffer * rtrReply;

/* Place CAN Module in configuration mode.*/

C1CONbits.REQOP = 4;
while(C1CONbits.OPMOD != 4);

/* Configure FIFO1 for transmit operation, 4 message buffers and enable */
/* Auto Remote Transmit. */

C1FIFOCON1SET = 0x00000080; /* Set the TXEN bit */
C1FIFOCON1SET = 0x00000004; /* Enable RTR */
C1FIFOCON1bits.FSIZE = 4;

/* Configure a filter to accept a message with SID = 0x100. Refer to */
/* 34.8 "CAN Message Filtering" for more details on configuring */
/* filters. In this case, Filter 0 and Mask0 are used. */

C1FLTCON0bits.FSELO = 1; /* Point to FIFO1 */
C1FLTCON0bits.MSELO = 0; /* Select Mask 0 */

C1RXF0bits.SID = 0x100; /* Configure Filter 0. */
C1RXF0bits.EXID = 0;

C1RXM0bits.SID = 0x1FF; /* Configure Mask 0. */
C1RXM0bits.MIDE = 1;

C1FLTCON0SET = 0x00000080; /* Enable the filter. */

/* Assign FIFO memory to CAN module */
C1FIFOBA = KVA_TO_PA(CANFIFO);

/* Place CAN Module in normal mode.*/

C1CONbits.REQOP = 0;
while(C1CONbits.OPMOD != 0);

/* Form the remote reply message. */

rtrReply = (CANMessageBuffer *) (PA_TO_KVA1(C1FIFOA1));
rtrReply->messageWord[0] = 0;
rtrReply->messageWord[1] = 0;
rtrReply->messageWord[2] = 0;
rtrReply->messageWord[3] = 0;

rtrReply->CMSGSID.SID = 0x100; /* CMSGSID */
rtrReply->CMSGEID.IDE = 0;
rtrReply->CMSGEID.DLC = 0x4;
rtrReply->messageWord[2] = 0x12BC1245; /* CMSGDAT0 */

C1FIFOCON1bits.UINC = 1;

/* FIFO is now ready to respond to RTR. */
```

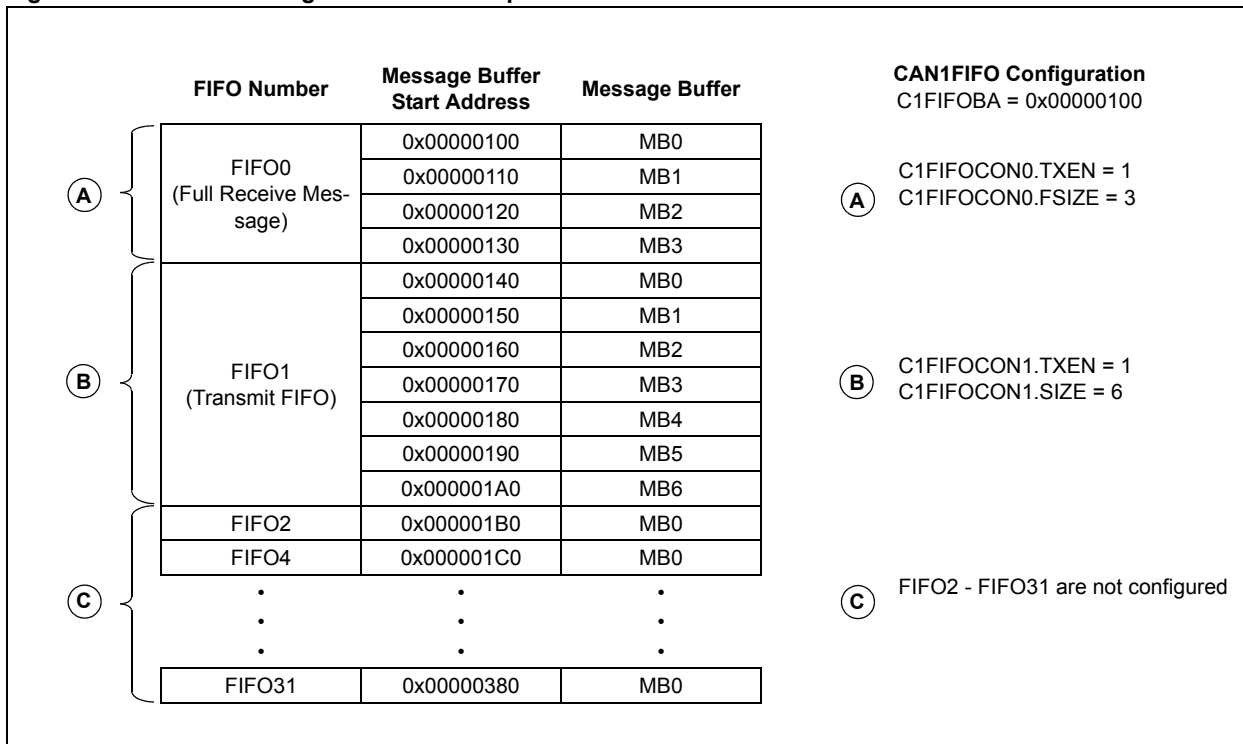


# Section 34. Controller Area Network (CAN)

## 34.7.6 Example Message Transmission FIFO Behavior

Consider an example user application that uses two FIFOs, FIFO0 and FIFO1, of the CAN1 module. FIFO0 has four message buffers configured for CAN message reception. FIFO1 has seven message buffers and is configured for message transmission. FIFO2 through FIFO31 are left in a default state. The CAN message buffer starts at physical address 0x00000100. This value is loaded in the C1FIFоба register. Figure 34-16 shows this application configuration.

Figure 34-16: FIFO Configuration for Example FIFO Behavior



The start address of FIFO1 can be calculated from the base address of the CAN message buffer (0x00000100) and the byte size of FIFO0 (4 message buffers \* 16 = 0x40). The physical start address of FIFO1 is 0x00000140. The below section focuses on the transmit FIFO1.

Figure 34-17 illustrates the case where FIFO1 of CAN1 module is empty and no messages have been written. The FIFO Transmit Not Full Interrupt Flag (TXNFULLIF), FIFO Transmit Half Empty interrupt flag (TXHALFIF) and the FIFO Transmit Empty Interrupt Flag (TXEMPTYIF) are set.

Figure 34-17: FIFO1 at Start

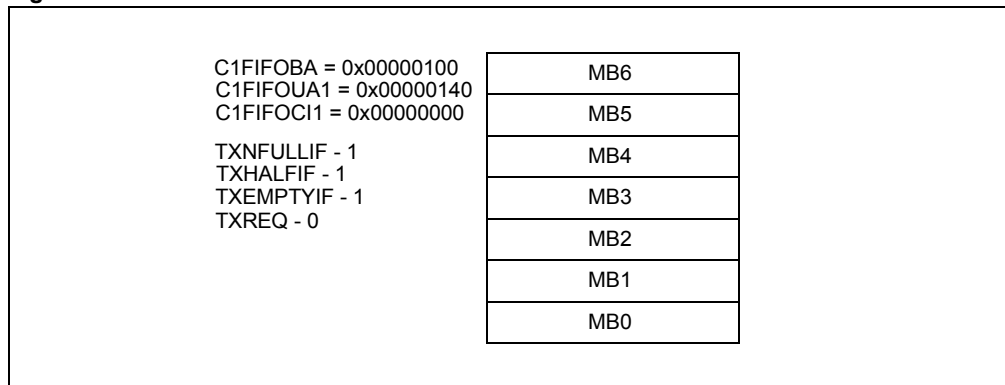


Figure 34-18 illustrates FIFO1 after the first message has been loaded to the FIFO. Note that the first message buffer (MB) in FIFO1, MB0, contains data. The user application sets the UINC bit (C1FIFOCON1<13>), which causes the FIFO head (C1FIFOUA1) to advance and point to the next empty message buffer, MB1. The TXEMPTYIF flag is cleared since the FIFO is not empty. The user application at this point has requested a transmission by setting the TXREQ bit (C1FIFOCON1<3>).

**Figure 34-18: FIFO1 - First Write**

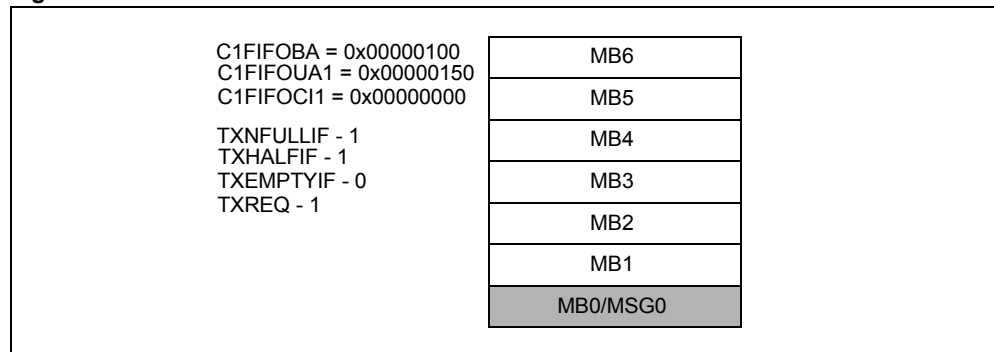


Figure 34-19 illustrates FIFO1 after the first message has been transmitted from MB0. TXREQ has been cleared and TXEMPTYIF is set once again. Note that the CAN module Message Index register, C1FIFOC11, now points to the second message buffer, MB1 at address 0x00000150.

**Figure 34-19: FIFO1 - First Message Transmitted**

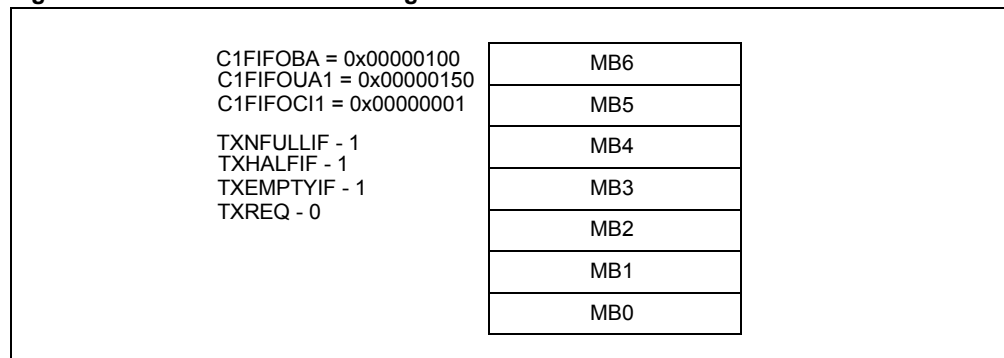
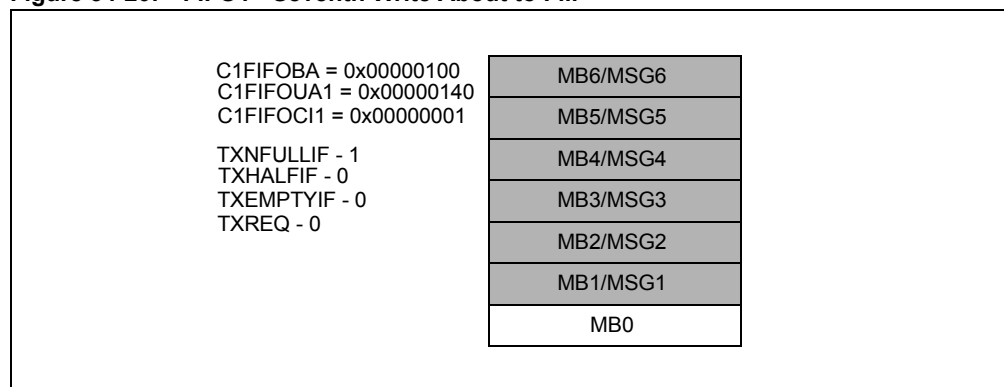


Figure 34-20 illustrates the FIFO after six more messages have been loaded into MB1 through MB6. The user application will have read C1FIFOUA1 when loading the messages to get the address location of the next message buffer to write to. The TXHALFIF flag is cleared and the user application has not requested the data be transmitted (TXREQ = 0).

**Figure 34-20: FIFO1 - Seventh Write About to Fill**



## Section 34. Controller Area Network (CAN)

Figure 34-21 illustrates the FIFO after an eighth message has been loaded, this time into MB0. The TXNFULLIF and TXHALFIF flags are both clear. The FIFO head now points to MB1. The FIFO at this stage is full. The user application has also requested that the message be transmitted (TXREQ = 1).

**Figure 34-21: FIFO1 - Eighth Write Buffer Full**

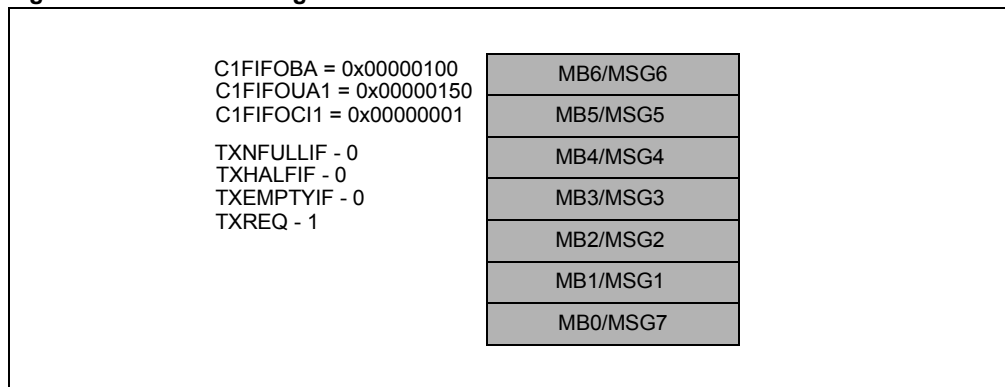
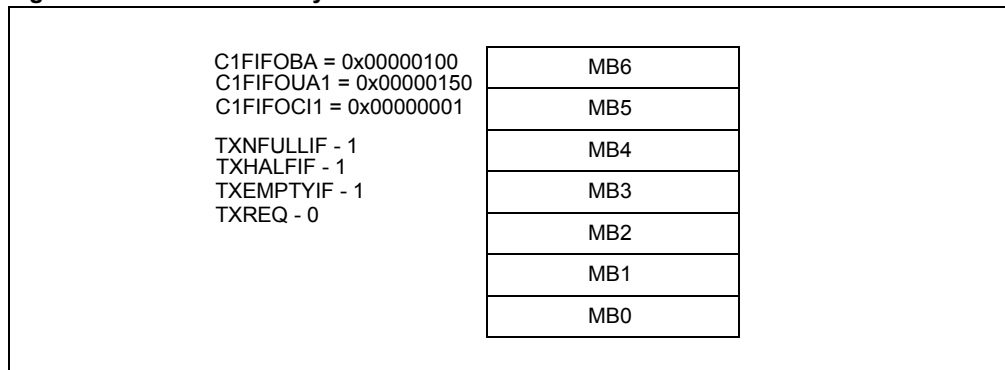


Figure 34-22 illustrates the FIFO empty once again. The CAN module has transmitted all of the seven messages in the FIFO. TXEMPTYIF is set once again, and TXREQ has been cleared by hardware. At this stage, the C1FOC11 has wrapped completely around and once again points at MB1.

**Figure 34-22: FIFO1 - Fully Transmitted**



## 34.8 CAN MESSAGE FILTERING

The CAN network is a broadcast type of network. A message transmitted by one node is received by all nodes in the network. Individual CAN nodes require a filtering mechanism to receive messages of interest. This filtering mechanism is provided by the CAN message acceptance filters and mask registers. Filtering is performed on the ID field of the CAN message.

The PIC32 CAN module has a total of 32 acceptance filters and four mask registers. The user application configures the specific filter to receive a message with a given identifier by setting a filter to match the identifier of the message to be received.

Each filter is controlled by its own CAN Filter Control register (CiFLTCONn). This register has the following bits for each filter:

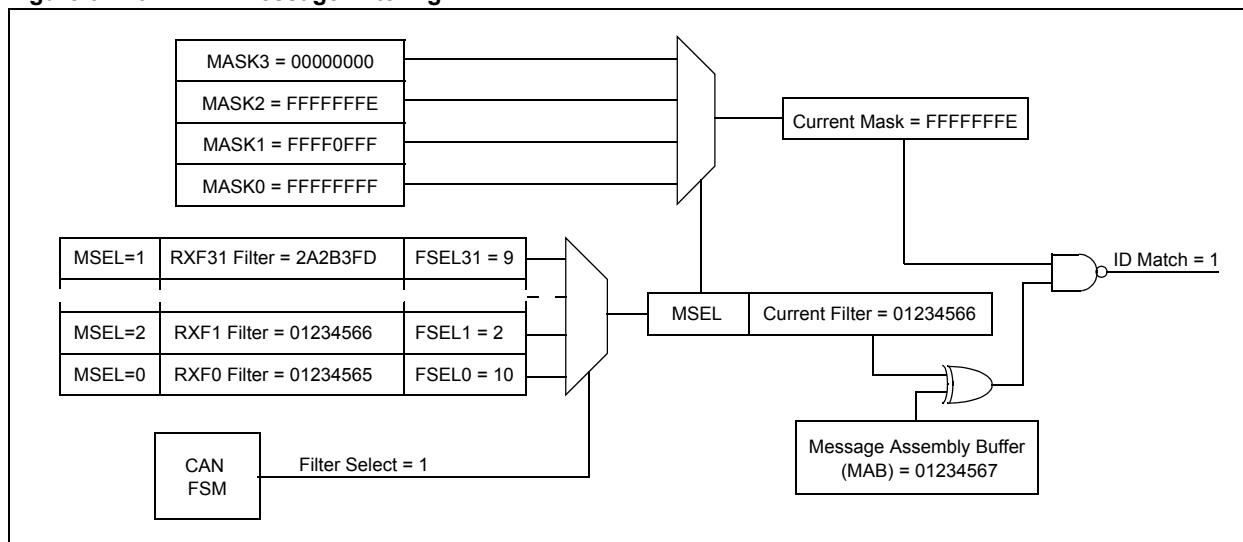
- FLTENn (n = 0 through 31) - Filter Enable bit enables/disables the filter
- MSELn<1:0> (n = 0 through 31) - Mask Select defines which mask is applied in the filter comparison
- FSELn<4:0> (n = 0 through 31) - Buffer Pointer defines the destination FIFO of a message whose ID matches the filter

As messages are received by the CAN module, the message identifier is compared with the corresponding bits in the filters. If the identifier matches the filter configured by the user, the message will be stored into the FIFO pointed to by the FIFO Selection (FSELn<4:0>) bits in the CAN Filter Control registers (CiFLTCONn).

The acceptance masks can be used to ignore the selected bits of the identifier as they are received. These bits will not be compared with the bits in the filter as the message is received. For example, if the user would like to receive all of the messages with identifiers 0, 1, 2 and 3, the user would mask out the lower two bits of the identifier. There are four mask registers available. A filter can have any one of the four masks associated with it.

Figure 34-23 illustrates a schematic representation of the CAN message filtering. After a message is received, the CAN module loops through all of the filters. As each filter is selected, the appropriate mask is also selected using the MSELn<1:0> bits. The message in the Message Assembly Buffer (MAB) is compared to the filter (XOR). Any bits that should be excluded from the filtering is excluded using the selected mask. If all the conditions match, an ID match occurs and the CAN module will move the contents of the MAB into the appropriate FIFO, pointed to by the FSELn<4:0> bits (CiFLTCONn<4:0>).

**Figure 34-23: CAN Message Filtering**



In the example illustrated in Figure 34-23, the received message has an ID of 01234567 and the closest match is Filter 1, which has an ID of 01234566. Note that these two match completely except for the Least Significant bit (LSb). A match still occurs because the selected mask (Mask 2) is set to ignore the LSb of the message. Since Filter 2 has its FSELn bits set to 2, the CAN module will then store the message in FIFO2.

### 34.8.1 Enabling and Modifying Filters

Filters can be turned ON or OFF using the Filter Enable (FLTENn) bit in the CiFLTCOn register. Clearing the bit turns the appropriate filter Off and setting it turns the filter on. Filters can be modified when the CAN module is in Configuration mode (OPMOD<2:0> = 100), Normal Operation mode (OPMOD<2:0> = 000) or Disable mode (OPMOD<2:0> = 001).

The CAN module will hold off disabling a filter when a received message is being processed. The FLTENn bit will remain set until the filtering is complete. If a message hits this filter during filtering, the message received will be written to the buffer pointed to by the filter. The filter is disabled after the message has been processed.

The user application should poll the FLTENn bit to ensure that the filter has been disabled before modifying the filter. Writes to the filter registers are blocked when the filter is enabled.

### 34.8.2 Extended and Standard Identifiers

A CAN message can have an 11-bit SID or 29-bit EID. The SID<10:0> bits and the EID<17:0> bits in the CAN Acceptance Filter Mask n registers (CiRXFn) should be configured to match the SID, or SID and EID of the desired message. Setting the Extended Identifier Enable (EXID) bit (CiRXFn<19>) will enable the filter to match messages with EIDs. Clearing the EXID bit (CiRXFn<19>) will enable the filter to match messages with SIDs. If the mask Identifier Receive Mode (MIDE) bit (CiRXMn<19>) is clear, this type of message will be ignored and all message types that match the filter will be accepted.

### 34.8.3 Acceptance Mask

There are four dedicated mask registers in the CAN module, Mask Register 0, 1, 2 and 3. Any filter can select one of four mask options:

- Mask Register 0
- Mask Register 1
- Mask Register 2
- Mask Register 3

Selection of the mask for an acceptance filter is controlled by the MSELm<1:0> bits corresponding to the filter in the appropriate CiFLTCOn register. The mask register cannot be modified when the filter using the mask is enabled. The filter should be disabled before modifying the mask. If no masking is desired on a given filter, then corresponding mask bits should be set to one. This will cause the filtering logic to consider all filter bits while performing the filtering operation. [Table 34-3](#) shows a truth table for various combinations of IDE bit in the CAN Message, EXID bit (CiRXFn<19>) and the MIDE bit (CiRXMn<19>).

**Note:** The CiRXMn register can only be modified when the CAN module is in Configuration mode.

**Table 34-3: Standard/Extended Message Reception Truth Table**

Message IDE bit	Filter EXID bit	Mask MIDE bit	Comment	Message Accepted
0	0	1	Standard message, filter specifies standard only, SID matches	Yes
			Standard message, filter specifies standard only, SID does not match	No
1	1	1	Extended message, filter specifies extended only, SID and EID match	Yes
			Extended message, filter specifies extended only, either SID or EID do not match	No
1	0	1	Extended message, filter specifies standard only	No
0	1	1	Standard message, filter specifies extended only	No
0	x	0	Standard message, filter specifies ignoring type, SID matches	Yes
			Standard message, filter specifies ignoring type, SID does not match	No
1	x	0	Extended message, filter specifies ignoring type, SID and EID match	Yes
			Extended message, filter specifies ignoring type, either SID or EID do not match	No

**Legend:** x = don't care

## 34.8.4 Buffer Pointer

Associated with the acceptance filter's control register are the FSELn<4:0> bits (CiFLTCOnn<4:0>). This serves as an address pointer for the corresponding filter. As the identifier comes in and if a match occurs, the FSELn output corresponding to that filter is enabled and latched into an address pointer for the receive FIFO memory. After the message has been assembled in the MAB, it is written into the selected FIFO. The address of the filter that matched the message is loaded into the Filter Hit Number (FILHIT<4:0>) bits (CIVEC<12:8>) within the receive buffer.

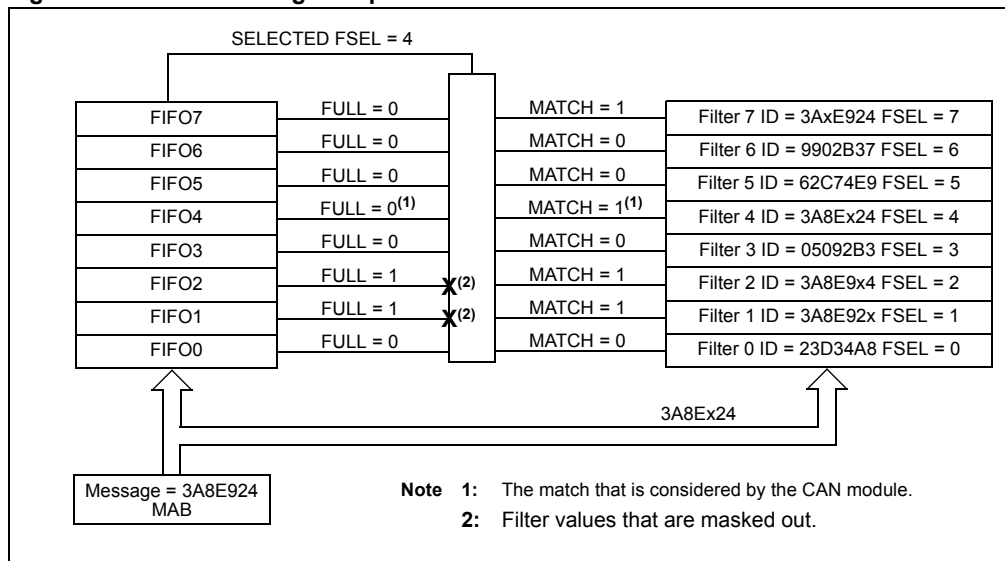
The use of a buffer pointer means that any filter can be made to point to any FIFO, and any FIFO can have multiple filters pointed to it. An example of how this can be used in a system would be to have all receive messages with a low priority be placed in one large FIFO, which is serviced infrequently by the CPU, and high priority messages placed into a second FIFO, which is serviced as the messages arrive.

## 34.8.5 Handling Messages with the Same ID

With the message filtering and masking options available in the CAN module, it is possible that one message may match more than one filter. When two or more filters match a message received and point to different receive buffer registers, only one receive buffer register will be loaded. The receive buffer register to be loaded will be determined by the filter number and buffer availability. The lower number filter will always have priority over a higher number filter. For example, Filter 0 will always take precedence over Filter 1. If the receive buffer register that Filter 0 points to is full, then the message will be loaded into the next available FIFO associated to a matching filter.

Figure 34-24 illustrates an example of the prioritization of multiple filter hits. The message in the MAB matches Filters 1, 2, 4 and 7. Filter 1 has the highest natural priority, but the associated FIFO is full, as is Filter 2. Filter 4 has the next highest priority and has room. The message is stored in FIFO4.

**Figure 34-24: Prioritizing Multiple Filter Hits**



### 34.8.6 Configuring a Filter

The following steps should be followed for configuring a filter for SID CAN messages:

1. Update the SID<10:0> bits (CiRXFn<0:0>) (n is the desired filter) with the SID of the required CAN message.
2. Clear the EXID bit (CiRXFn<19>). This will cause the filter to match SID messages only.
3. Create a mask for the filter. Load the appropriate value in the CiRXMn register. Set the MIDE bit (CiRXMn<19>) to match only SID messages
4. Identify the CiFLTCOnn register which controls the selected filter. Set the value of the FSELn<4:0> bits (CiFLTCOnn<4:0>) to point to the receive FIFO. Set the MSELn bits to use the mask configured in step 3.
5. Enable the filter by setting the FLTENn bit in the CiFLTCOnn register.

Example 34-11 shows a code example for configuring the CAN module message acceptance filter for SIDs.

#### Example 34-11: Configuring a Filter to Match a SID CAN Message

```

/* This code example shows how to configure a filter to match a Standard */
/* Identifier (SID) CAN message. */

/* In this example, Filter 3 is set up to accept messages with a SID range */
/* of 0x100 to 103. Accepted messages will be stored in FIFO5. Mask 1 is */
/* used to implement the filter address range. */

C1FLTCOn0bits.FSEL3 = 5; /* Store messages in FIFO5 */
C1FLTCOn0bits.MSEL3 = 1; /* Use Mask 1 */

C1RXF3bits.SID = 0x100; /* Filter 3 SID */
C1RXF3bits.EXID = 0; /* Filter only SID messages */

C1RXM1bits.SID = 0x7FC; /* Ignore last 2 bits in comparison */
C1RXM1bits.MIDE = 1; /* Match only message types. */

C1FLTCOn0bits.FLTEN3 = 1; /* Enable the filter */

/* Filter is now configured. */
    
```

The following steps should be followed for configuring a filter for EID CAN messages:

1. Update the SID<10:0> bits (CiRXFn<31:21>) (where n is the desired filter) with the SID of the required CAN message. Update the EID<17:0> bits (CiRXFn <17:0>) with the EID of the required CAN message
2. Set the EXID bit (CiRXFn<19>) register. This will cause the filter to match EID messages only.
3. Create a mask for the filter. Load the appropriate value in the CiRXMn register. Set the MIDE bit (CiRXMn<19>) to match only EID messages.
4. Identify the CiFLTCONn register which controls the selected filter. Set the value of the FSELn<4:0> bits (CiFLTCONn<4:0>) to point to the receive FIFO. Set the MSELn bits to use the mask configured in step 3.
5. Enable the filter by setting the FLTENn bit in the CiFLTCONn register.

Example 34-12 shows a code example for configuring the CAN module message acceptance filter for EIDs.

### Example 34-12: Configuring a Filter to Match an EID CAN Message

```
/* This code example shows how to configure a filter to match an */
/* Extended Identifier CAN message. */

/* In this example, Filter 3 is set up to accept messages with SID = 0x53 */
/* and EID = 0x1C498. Accepted message will be stored in FIFO5. Mask 1 is */
/* used. */

C1FLTCON0bits.FSEL3 = 5;      /* Store messages in FIFO5 */
C1FLTCON0bits.MSEL3 = 1;     /* Use Mask 1 */

C1RXF3bits.SID = 0x100;      /* Filter 3 SID */
C1RXF3bits.EID = 0x1C498;    /* Filter 3 EID */
C1RXF3bits.EXID = 1;        /* Filter only SID messages */

C1RXM1bits.SID = 0x7FF;      /* Consider all bits in */
C1RXM1bits.EID = 0x3FFFF;    /* in the filter comparison. */
C1RXM1bits.MIDE = 1;        /* Match only message types. */

C1FLTCON0bits.FLTEN3 = 1;    /* Enable the filter */

/* Filter is now configured. */
```



# Section 34. Controller Area Network (CAN)

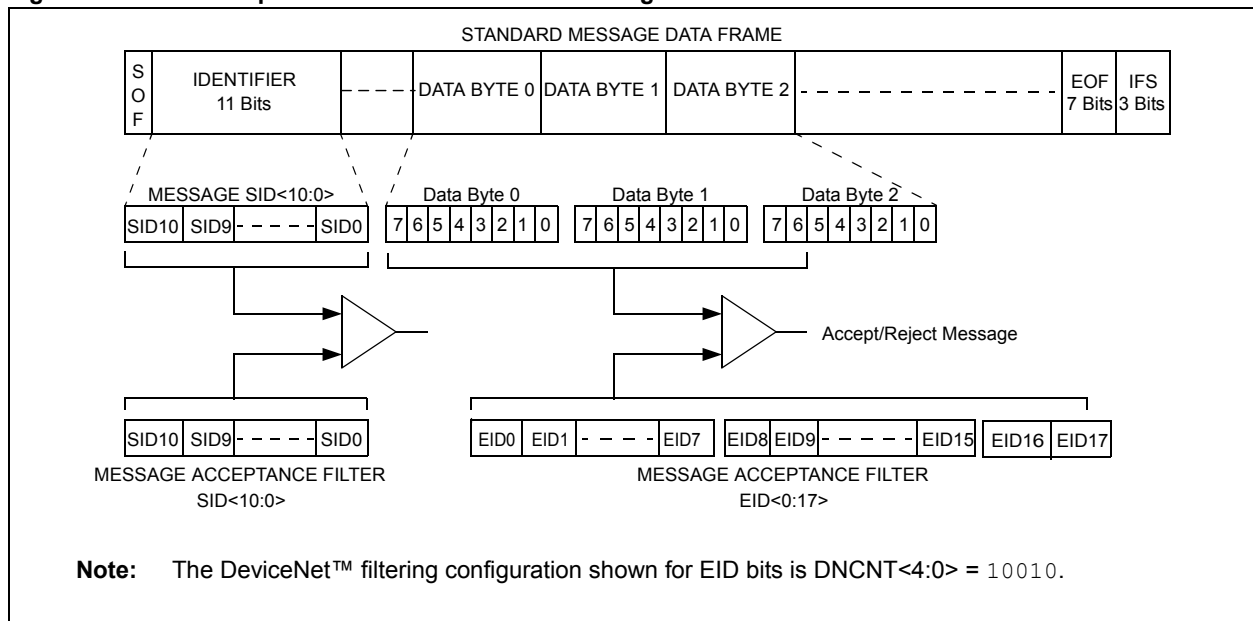
## 34.8.7 DeviceNet™ Filtering

The DeviceNet filtering feature is based on the CAN Specification 2.0A, in which up to 18 bits of the data field can be compared with the EID of the message acceptance filter in addition to the SID.

The DeviceNet feature is enabled or disabled by the DeviceNet Filter Bit Number (DNCNT<4:0>) control bits in the CAN Control register (CiCON<4:0>). The value specified in the DNCNT field determines the number of data bits to be used for comparison with the EID bits of the message acceptance filter. If the DNCNT<4:0> bits (CiCON<4:0>) are cleared, the DeviceNet feature is disabled.

For a message to be accepted, the 11-bit SID must match the SID<10:0> bits in the message acceptance filter and the first 'n' data bits in the message should match the EID<17:0> bits in the message acceptance filter. For example, as shown in Figure 34-25, the first 18 data bits of the received message data payload are compared with the corresponding EID bits of the message acceptance filter (CiRXFn<17:0>). The IDE bit of the received message must be '0'.

**Figure 34-25: CAN Operation with DeviceNet™ Filtering**



## 34.8.7.1 FILTER COMPARISONS

Table 34-4 shows the filter comparisons configured by the DNCNT<4:0> control bits (CiCON<4:0>). For example, if DNCNT<4:0> = 00011, only a message in which the 11-bit SID matches the SID acceptance filter (SID<10:0>) and bits 7, 6 and 5 of Data Byte 0 match the EID filter (EID<0:2>) is accepted.

**Table 34-4: DeviceNet™ Filter Bit Configurations**

DeviceNet™ Filter Configuration (DNCNT<4:0>)	Received Message Data Bits to be Compared (Byte<bits>)	EID Bits Used for Acceptance Filter
00000	No comparison	No comparison
00001	Data Byte 0<7>	EID<0>
00010	Data Byte 0<7:6>	EID<1:0>
00011	Data Byte 0<7:5>	EID<2:0>
00100	Data Byte 0<7:4>	EID<3:0>
00101	Data Byte 0<7:3>	EID<4:0>
00110	Data Byte 0<7:2>	EID<5:0>
00111	Data Byte 0<7:1>	EID<6:0>
01000	Data Byte 0<7:0>	EID<7:0>
01001	Data Byte 0<7:0> and Data Byte 1<7>	EID<8:0>
01010	Data Byte 0<7:0> and Data Byte 1<7:6>	EID<9:0>
01011	Data Byte 0<7:0> and Data Byte 1<7:5>	EID<10:0>
01100	Data Byte 0<7:0> and Data Byte 1<7:4>	EID<11:0>
01101	Data Byte 0<7:0> and Data Byte 1<7:3>	EID<12:0>
01110	Data Byte 0<7:0> and Data Byte 1<7:2>	EID<13:0>
01111	Data Byte 0<7:0> and Data Byte 1<7:1>	EID<14:0>
10000	Data Byte 0<7:0> and Data Byte 1<7:0>	EID<15:0>
10001	Byte 0<7:0> and Byte 1<7:0> and Byte 2<7>	EID<16:0>
10010	Byte 0<7:0> and Byte 1<7:0> and Byte 2<7:6>	EID<17:0>
10011 to 11111	Invalid Selection	Invalid Selection

## 34.8.7.2 SPECIAL CASES

There may be special cases when the message contains fewer data bits than are called for by the DeviceNet filter configuration.

- **Case 1** – If DNCNT <4:0> is greater than 18, indicating that the user application selected a number of bits greater than the total number of EID bits, the filter comparison terminates with the eighteenth bit of the data (bit 6 of data byte 2). If the SID and all 18 data bits match, the message is accepted.
- **Case 2** – If DNCNT<4:0> is greater than 16, and the received message Data Length Code (DLC) is 2 (indicating a payload of two data bytes), the filter comparison terminates with the sixteenth bit of data (bit 0 of data byte 1). If the SID and all 16 bits match, the message is accepted.
- **Case 3** – If DNCNT<4:0> is greater than 8, and the received message has DLC = 1 (indicating a payload of one data byte), the filter comparison terminates with the eighth bit of data (bit 0 of data byte 0). If the SID and all 8 bits match, the message is accepted.
- **Case 4** – If DNCNT<4:0> is greater than 0, and the received message has DLC = 0, indicating no data payload, the filter comparison terminates with the SID. If the SID matches, the message is accepted.

# Section 34. Controller Area Network (CAN)

## 34.9 RECEIVING A CAN MESSAGE

The CAN module continually monitors messages on the CAN bus. As messages are received by the CAN module, the message identifier is compared to the filter/mask combinations that are currently configured. If a match occurs, the CAN module will store the message in the FIFO pointed to by the FSELn<4:0> bits (CiFLTCONn<4:0>). Once the message has been received and stored in the FIFO, the following bits will be updated:

- The CFIFOCIn<4:0> bits in the CAN Module Message Index register (CiFIFOCIn<4:0>) will be updated
- The Receive FIFO Overflow interrupt flag (RXOVFLIF), Receive FIFO Full interrupt flag (RXFULLIF), Receive FIFO Not Empty interrupt flag (RXEMPTYIF), and the Receive FIFO Half Full interrupt flag (RXHALFIF) in the CiFIFOINTn register will be updated
- The FIFO Interrupt Pending Flag (FIFOIPn<31:0>) bits in the CAN FIFO Status register (CiFSTAT<31:0>) will be updated
- An interrupt will be generated, if the interrupts are enabled

The ICODE<6:0> bits (CiVEC<6:0>) and the FILHIT<4:0> bits (CiVEC<12:8>) will also be updated.

**Note:** The user application must enable at least one message acceptance filter and one mask register in order to receive messages.

The accepted CAN message is stored in the message buffer using the format shown in Table 34-5. The CAN module uses the formatting as illustrated in Figure 34-26 through Figure 34-29.

**Table 34-5: Receive Message Format as Stored in RAM - CiCON.CANCAP = 1, CFIFOCON.DONLY = 0**

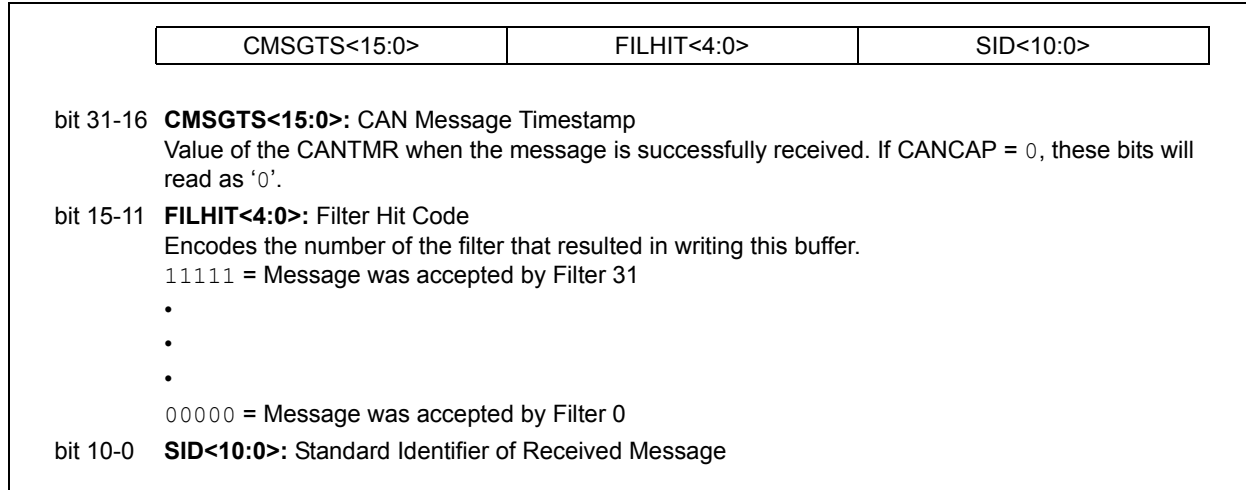
Address Offset	Name	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
0x00	MSGSID	31:24 MSGSIDS<15:8>							
		23:16 MSGSIDS<7:0>							
		15:8 FILHIT<4:0>				SID<10:8>			
		7:0 SID<7:0>							
0x04	MSGGEID	31:24 ---		SRR	IDE	EID<17:14>			
		23:16 EID<13:6>							
		15:8 EID<5:0>						RTR	RB1
		7:0 ---		---		RB0	DLC<3:0>		
0x08	MSGDATA0	31:24 Receive Buffer Data Byte 3							
		23:16 Receive Buffer Data Byte 2							
		15:8 Receive Buffer Data Byte 1							
		7:0 Receive Buffer Data Byte 0							
0x0C	MSGDATA1	31:24 Receive Buffer Data Byte 7							
		23:16 Receive Buffer Data Byte 6							
		15:8 Receive Buffer Data Byte 5							
		7:0 Receive Buffer Data Byte 4							

**Legend:** Shaded bits read as '0'.

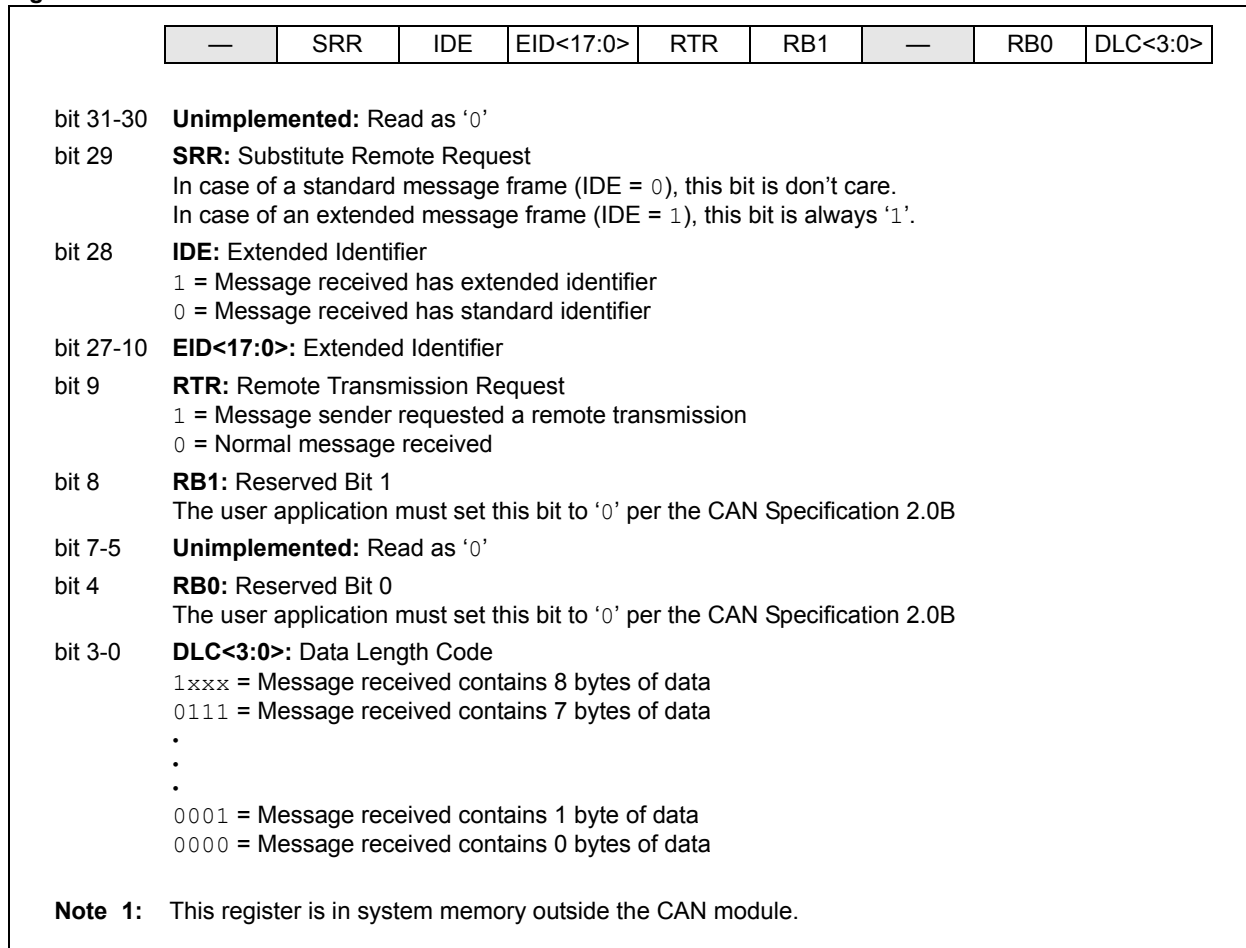
**Note 1:** The CAN receive message is stored in device RAM and does not have SET/CLR/INV registers associated with it.

# PIC32 Family Reference Manual

**Figure 34-26: Format of CMSGSID**

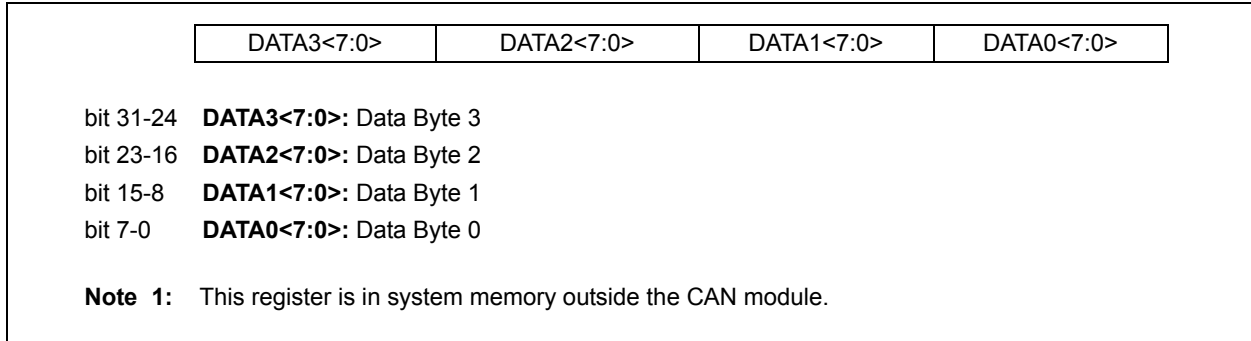


**Figure 34-27: Format of CMSGEID**

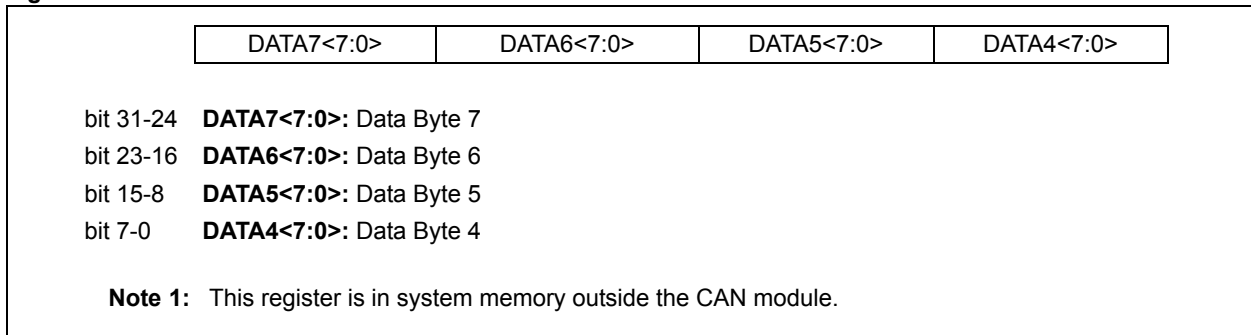


## Section 34. Controller Area Network (CAN)

**Figure 34-28: Format of CMSGDATA0**



**Figure 34-29: Format of CMSGDATA1**



The code in [Example 34-13](#) shows an example data structure for implementing a CAN full receive message buffer. An example of using this structure is provided in [Example 34-14](#).

## Example 34-13: Implementing a CAN Full Receive Message Buffer in Memory

```
/* This code snippet shows an example of data structure to implement a CAN */
/* full receive message buffer.*/
/* Define the sub-components of the data structure as specified in Table 34-5 */
/* Create a CMSGSID data type. */
typedef struct
{
    unsigned SID:11;
    unsigned FILHIT:5;
    unsigned CMSGTS:16;
}rxcmsgsid;

/* Create a CMSGEID data type. */
typedef struct
{
    unsigned DLC:4;
    unsigned RB0:1;
    unsigned :3;
    unsigned RB1:1;
    unsigned RTR:1;
    unsigned EID:18;
    unsigned IDE:1;
    unsigned SRR:1;
    unsigned :2;
}rxcmsgeid;

/* Create a CMSGDATA0 data type. */
typedef struct
{
    unsigned Byte0:8;
    unsigned Byte1:8;
    unsigned Byte2:8;
    unsigned Byte3:8;
}rxcmsgdata0;

/* Create a CMSGDATA1 data type. */
typedef struct
{
    unsigned Byte4:8;
    unsigned Byte5:8;
    unsigned Byte6:8;
    unsigned Byte7:8;
}rxcmsgdatal;

/* This is the main data structure. */
typedef union uCANRxMessageBuffer {
    struct
    {
        rxcmsgsid CMSGSID;
        rxcmsgeid CMSGEID;
        rxcmsgdata0 CMSGDATA0;
        rxcmsgdatal CMSGDATA1;
    };
    int messageWord[4];
}CANRxMessageBuffer;
```

## Example 34-14: Example Usage of the Data Structure

```
/* Example usage of code provided in Example 34-13. */
CANRxMessageBuffer * buffer;

/* When a message have been received and read, the individual fields of */
/* the received message can be queried as such. */
buffer = (CANRxMessageBuffer *) (PA_TO_KVA1(C1FIFOUA1));

if(buffer->CMSGEID.DLC == 4)
{
    /* If the length of the received message is 4 then do something. */
}
```

# Section 34. Controller Area Network (CAN)

## 34.9.1 Data-Only Receive Messages

The PIC32 CAN module provides a special receive mode where only the data payload section of the received message is stored in the message buffer. The CAN module will discard the identifier, the reserved bits and the DLC bits. The time stamp value is not stored even if time stamping is enabled. This mode can be enabled by setting the DONLY bit (C1FIFOCONn<12>).

Note that eight bytes of data are stored, regardless of the value of the DLC field in the CAN message. The unused bytes are filled with 0x00. One possible use of this mode is to concatenate messages whose data spans multiple messages. For example, transmitting a string across the CAN bus.

**Table 34-6: Data-Only Receive Message Format as Stored in RAM**

Address Offset	Name	Bit 31/23/15/7	Bit 30/22/14/6	Bit 29/21/13/5	Bit 28/20/12/4	Bit 27/19/11/3	Bit 26/18/10/2	Bit 25/17/9/1	Bit 24/16/8/0
0x00	CMSG0DATA0	31:24	Receive Buffer Data Byte 3						
		23:16	Receive Buffer Data Byte 2						
		15:8	Receive Buffer Data Byte 1						
		7:0	Receive Buffer Data Byte 0						
0x04	CMSG0DATA1	31:24	Receive Buffer Data Byte 7						
		23:16	Receive Buffer Data Byte 6						
		15:8	Receive Buffer Data Byte 5						
		7:0	Receive Buffer Data Byte 4						

**Note 1:** The messages are stored in system memory.

**Note 2:** As a result there are no SET/CLR/INV registers associated with these registers.

## 34.9.2 Processing a Received Message

The user application can either use the polling methods or use the CAN module interrupt to track message reception. The CAN module provides notification about different Receive FIFO events. The user application can be notified when the Receive FIFO is not empty, is half-full or is full. The user applications should read the Receive FIFO frequently to ensure that there is no FIFO overflow. Regardless of the technique used (polling or interrupts), the following steps can be used to read the message from the FIFO:

1. Read the value of the CiFIFOUn register. This provides a 32-bit physical address pointer to the receive message buffer that the user application should read.
2. In case of the Data-only receive message type, process two words from the message buffer.
3. In case of Full receive message type, process four words from the message buffer.
4. After processing the message buffer, set the UINC bit (C1FIFOCONn<13>).

Example 34-15 shows a code example of copying a CAN message.

**Example 34-15: Copying a Message from a Receive FIFO**

```

/* This code example shows how to process a message from a receive FIFO. */
/* In this example CAN1 and FIF01 are used. */

CANRxMessageBuffer rxMessage;
unsigned int * bufferToRead;

/* Check if there is a message available to read. */
if(C1FIFOINT1bits.RXEMPTYIF == 1)
{
    /* Get the address of the buffer to read */
    bufferToRead = PA_TO_KVAL(C1FIFOUA1);

    /* This is an example process (Copying the buffer). The application */
    /* could process this buffer in place. */
    bufferToRead[0] = rxMessage->messageWord[0];
    bufferToRead[1] = rxMessage->messageWord[1];
    bufferToRead[2] = rxMessage->messageWord[2];
    bufferToRead[3] = rxMessage->messageWord[3];

    /* Update the message buffer pointer. */
    C1FIFOCON1bits.UINC = 1;
}

```

## 34.9.3 Example Receive FIFO Behavior

Consider a user application where FIFO0 of the CAN1 module is configured as a receive message FIFO. Figure 34-30 illustrates the FIFO before the first message arrives and is placed in the FIFO. The CAN FIFO Base Address register (C1FIFOB) is set to 0x00000100. The received messages will be placed in device RAM (physical address 0x00000100). Note that the C1FIFOUA0 initially points to the start of the FIFO. The example tracks the Receive FIFO Overflow Interrupt Flag (RXOVFLIF), Receive FIFO Full Interrupt Flag (RXFULLIF), Receive FIFO Half Full Interrupt Flag (RXHALFIF) and Receive FIFO Not Empty Interrupt Flag (RXEMPTYIF). This configuration is illustrated in Figure 34-30.

**Figure 34-30: FIFO - At Start**

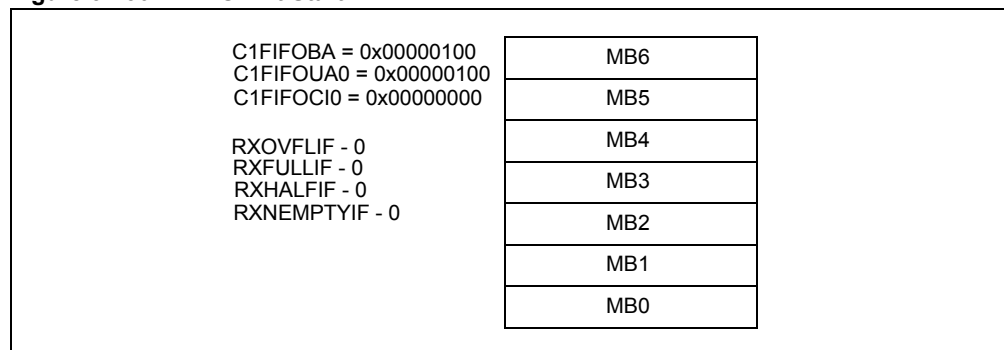


Figure 34-31 illustrates the FIFO after the first message has been received and written to the FIFO. Note that while the first message buffer in the FIFO (MB0) contains data, the C1FIFOUA0 still points to the base address of the FIFO because the user has not read this location. Note that the CAN module message index register (C1FIFOCIO) has incremented showing that the next received message will be placed in MB1. The RXEMPTYIF flag is set indicating that the FIFO is not empty.

**Figure 34-31: FIFO - First Write**

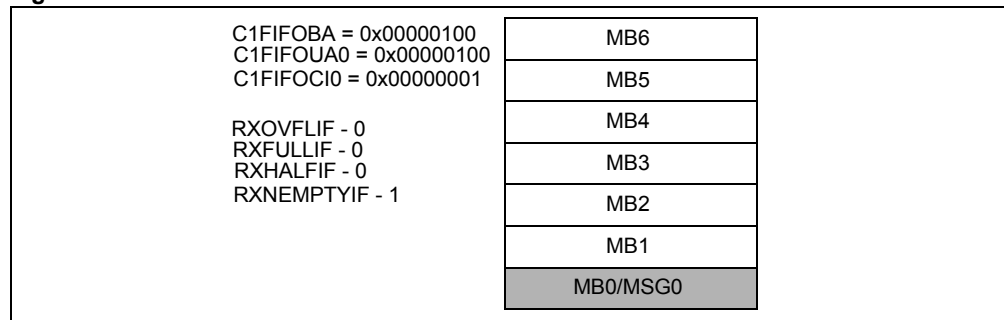
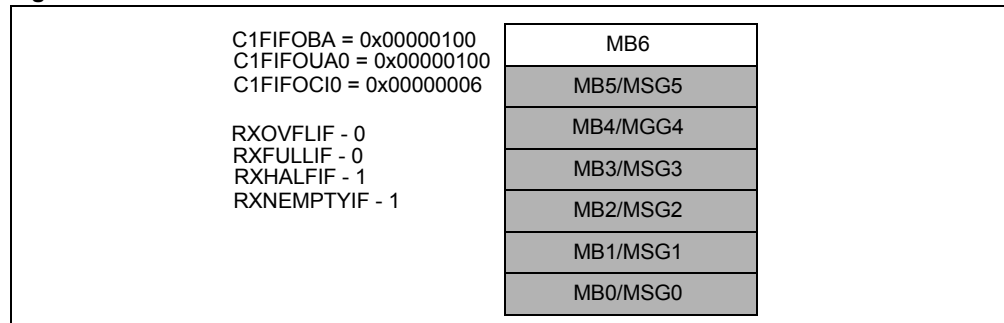


Figure 34-32 illustrates the FIFO after another five messages have been received as the C1FIFOUA0 has not been modified earlier. The Buffer Half Full flag (RXHALFIF) was set after the reception of the fourth message.

**Figure 34-32: FIFO - Sixth Write**





## Section 34. Controller Area Network (CAN)

Figure 34-33 illustrates the FIFO after the first message has been read from MB0. Once the user application has read this message, it should set the UINC bit (CiFIFOCONn<13>). This will cause the CFIFOUA0 register to change to 0x0000110 (which is the address of the next message buffer the user application must read).

**Figure 34-33: FIFO - First Read**

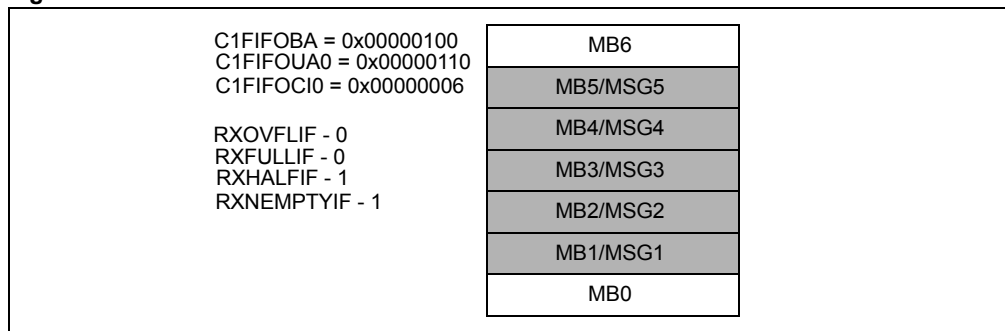


Figure 34-34 illustrates the FIFO after receiving a seventh message.

**Figure 34-34: FIFO - Seventh Write About to Fill**

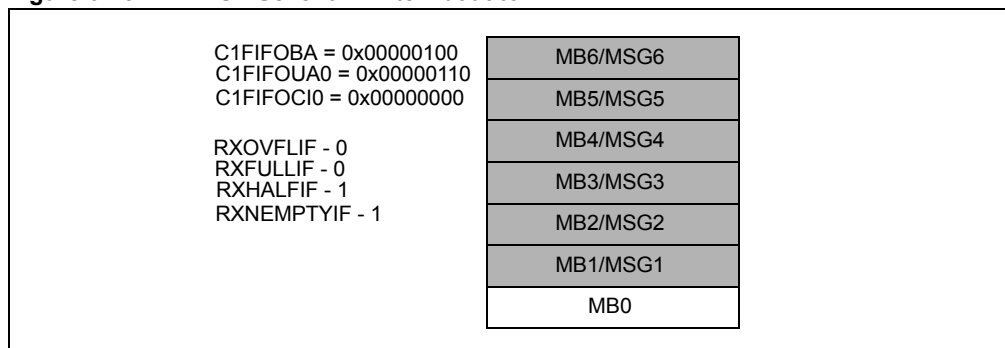


Figure 34-35 illustrates the FIFO in a full state. Note that the RXFULLIF flag is set.

**Figure 34-35: FIFO - Eighth Write FIFO Full**

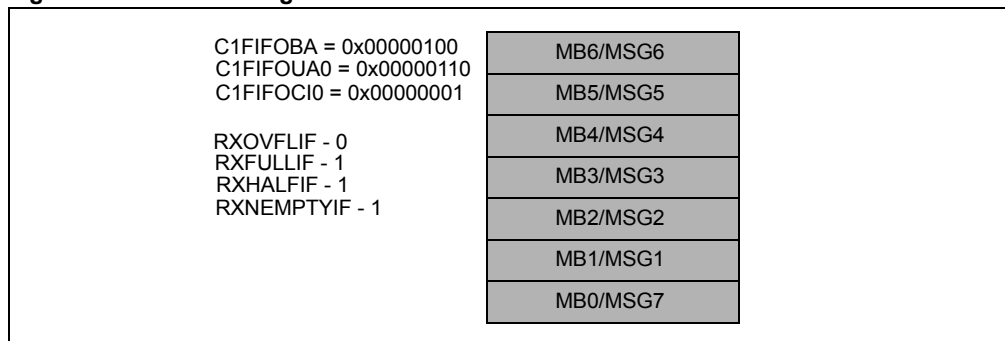
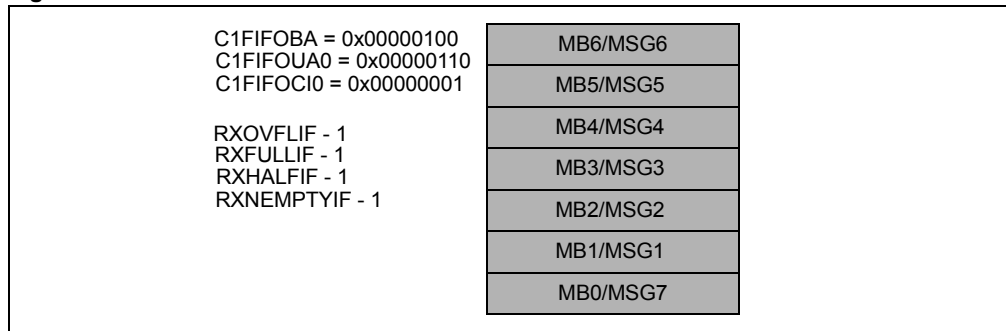


Figure 34-36 illustrates the FIFO receiving an additional message with the FIFO full, which overflows the FIFO. Note that the RXOVFLIF flag is set, the CAN index (CiCIFOCl) has not moved, and the message (MSG8) has been lost.

**Figure 34-36: FIFO - Ninth Write FIFO Overflow**



## 34.10 BIT TIMING

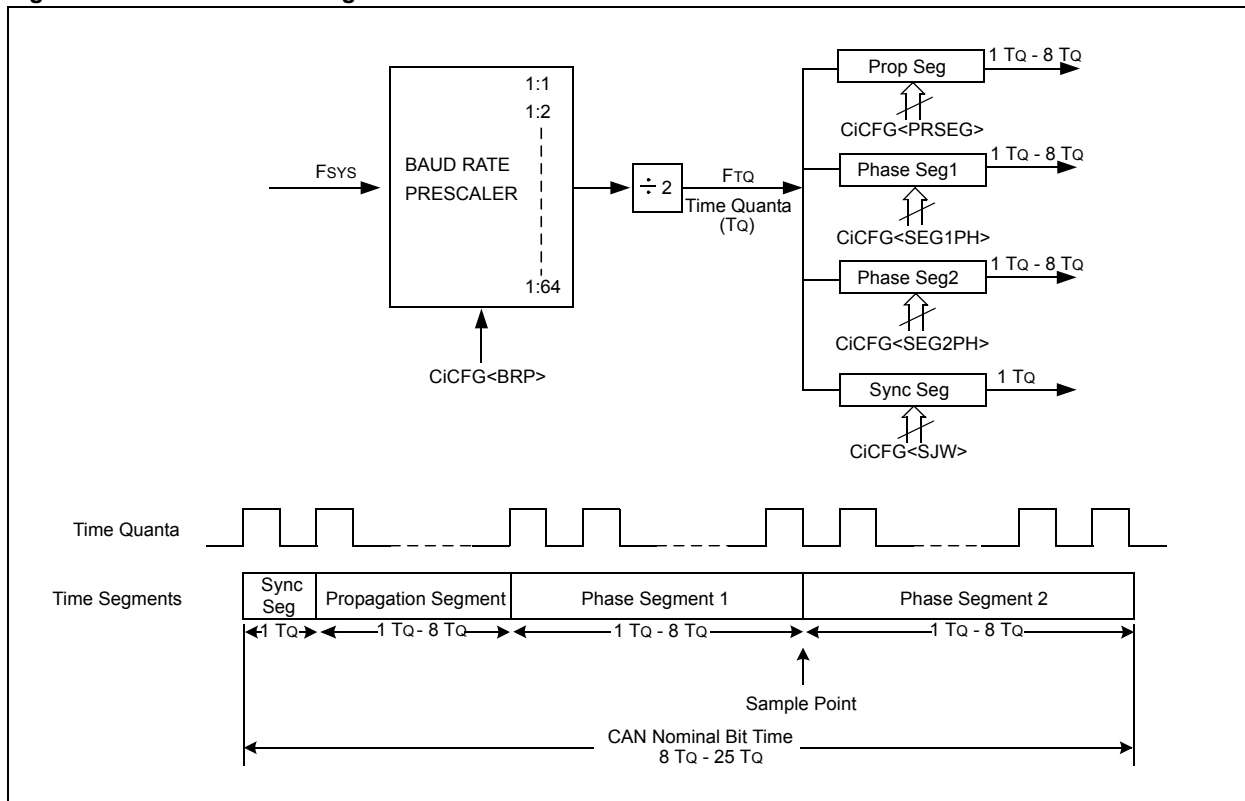
The nominal bit rate is the number of bits per second transmitted on the CAN bus.

$$\text{Nominal Bit Time} = 1 \div \text{Nominal Bit Rate}$$

There are four time segments in a bit time to compensate for any phase shifts due to oscillator drifts or propagation delays. These time segments do not overlap each other and are represented in terms of Time Quantum (T<sub>Q</sub>). One T<sub>Q</sub> is a fixed unit of time derived from the oscillator clock. The total number of time quanta in a nominal bit time must be programmed between 8 T<sub>Q</sub> and 25 T<sub>Q</sub>.

Figure 34-37 illustrates how the CAN Bit Time Quantum Frequency (F<sub>TQ</sub>) is obtained from the system clock and also how the different time segments are programmed.

Figure 34-37: CAN Bit Timing



### 34.10.1 Bit Segments

Each bit transmission time consists of four time segments:

- **Synchronization Segment** – This time segment synchronizes the different nodes connected on the CAN bus. A bit edge is expected to be within this segment. Based on CAN protocol, the Synchronization Segment is assumed to be 1 T<sub>Q</sub>.
- **Propagation Segment** – This time segment compensates for any time delay that may occur due to the bus line or due to the various transceivers connected on that bus.
- **Phase Segment 1** – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be lengthened during resynchronization to compensate for the phase shift.
- **Phase Segment 2** – This time segment compensates for errors that may occur due to phase shift in the edges. The time segment may be shortened during resynchronization to compensate for the phase shift. The Phase Segment 2 time can be configured to be either programmable or specified by the Phase Segment 1 time.

## 34.10.2 Sample Point

The sample point is the point in a CAN bit time interval where the sample is taken and the bus state is read and interpreted. It is situated between Phase Segment 1 and Phase Segment 2. The CAN bus can be sampled once or thrice at the sample point, as configured by the Sample CAN Bus Line (SAM) bit in the CAN Baud Rate Configuration register (CiCFG<14>).

- If CiCFG<14> = 1, the CAN bus is sampled three times at the sample point. The most common of the three samples determines the bit value.
- If CiCFG<14> = 0, the CAN bus is sampled only once at the sample point

## 34.10.3 Synchronization

Two types of synchronization are used: Hard Synchronization and Resynchronization. A Hard Synchronization occurs once at the SOF. Resynchronization occurs inside a frame.

- **Hard Synchronization** – takes place on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge
- **Resynchronization** – takes place when a bit edge does not occur within the Synchronization Segment in a message. One of the Phase Segments is shortened or lengthened by an amount that depends on the phase error in the signal. The maximum amount that can be used is determined by the SJW parameter (CiCFG<7:6>).

The length of Phase Segment 1 and Phase Segment 2 can be changed depending on oscillator tolerances of the transmitting node and receiving node. Resynchronization compensates for any phase shifts that may occur due to the different oscillators used by the transmitting and receiving nodes.

- **Bit Lengthening** – If the transmitting node in CAN has a slower oscillator than the receiving node, the next falling edge, and therefore, the sample point can be delayed by lengthening Phase Segment 1 in the bit time
- **Bit Shortening** – If the transmitting node in CAN has a faster oscillator than the receiving node, the next falling edge, and therefore, the sample point of the next bit can be reduced by shortening the Phase Segment 2 in the bit time
- **Synchronization Jump Width (SJW)** – The SJW<1:0> bits in CAN Baud Rate Configuration register (CiCFG<7:6>) determine the SJW by limiting the amount of lengthening or shortening that can be applied to the Phase Segment 1 and Phase Segment 2 time intervals. This segment should not be longer than Phase Segment 2 time. The width can be 1 T<sub>Q</sub> - 4 T<sub>Q</sub>.

## 34.10.4 CAN Bit Time Calculations

The steps that must be performed by the user application to configure the bit timing for the CAN module are described below along with examples.

### 34.10.4.1 STEP 1: CALCULATE THE CAN BIT TIME QUANTUM FREQUENCY (FTQ)

- Select the Baud Rate (F<sub>BAUD</sub>) for the CAN Bus.
- Select the number of time quanta in a bit time, based on your system requirements. Equation 34-1 shows the formula for computing FTQ.

#### Equation 34-1: CAN Bit Time Quantum Frequency (FTQ)

$$F_{TQ} = N \times F_{BAUD}$$

- Note 1:** The total number of time quanta in a nominal bit time must be programmed between 8 T<sub>Q</sub> and 25 T<sub>Q</sub>. Therefore, the FTQ is between 8 to 25 times the baud rate (F<sub>BAUD</sub>).
- 2:** Make sure that FTQ is an integer multiple of F<sub>BAUD</sub> to get the precise bit time. Otherwise, the oscillator input frequency or the F<sub>BAUD</sub> may need to be changed.

## Section 34. Controller Area Network (CAN)

### 34.10.4.2 STEP 2: CALCULATE THE BAUD RATE PRESCALER (BRP<5:0> (CiCFG<5:0>))

Equation 34-2 shows the formula for computing the baud rate prescaler.

#### Equation 34-2: Baud Rate Prescaler

$$CiCFG\langle BRP \rangle = (F_{sys} / (2 * FTQ)) - 1$$

### 34.10.4.3 STEP 3: SELECT THE INDIVIDUAL BIT TIME SEGMENTS

The Individual Bit Time Segments are selected using the CiCFG register:

Bit Time = Sync Segment + Propagation Segment + Phase Segment 1 + Phase Segment 2

**Note 1:** (Propagation Segment + Phase Segment 1) must be greater than or equal to the length of Phase Segment 2.

**2:** Phase Segment 2 must be greater than SJW.

Example 34-16 shows the procedure to calculate the FTQ.

#### Example 34-16: CAN Bit Timing Calculation Example

Step 1: Calculate the FTQ.

If FBAUD = 1 Mbps, and the number of time quanta 'N' = 10, then FTQ = 10 MHz

Step 2: Calculate the baud rate prescaler (assuming that the PIC32 device is running at 80 MHz, that is F<sub>sys</sub> = 80000000).

$$CiCFG\langle BRP \rangle = (80000000 / (2 * FTQ)) - 1 = 3$$

Step 3: Select the individual bit time segments.

Synchronization Segment = 1 Tq (constant)

Based on system characteristics, if Propagation Delay = 3 Tq, if the sample point is to be at 70% of Nominal Bit Time, then:

Phase Segment 2 = 30% of Nominal Bit Time = 3 Tq

Phase Segment 1 = 10 Tq - (1 Tq + 3 Tq + 3 Tq) = 3 Tq

Example 34-17 illustrates a code example for configuring the CAN bit timing code.

#### Example 34-17: Configuring the CAN Module to Obtain a Specific Bit Rate

```
/* This code example shows how to configure the CAN module to obtain a */
/* specific bit rate implementing the configuration shown in Example 34-16. */

/* Fsys = System Clock Frequency = 80000000;      */
/* Fbaud = CAN bit rate = 1000000;                */
/* N = Time Quanta (Tq) per bit = 10;             */
/* Prop Segment = 3 Tq                            */
/* Phase Seg 1 = 3 Tq                              */
/* Phase Seg 2 = 3 Tq                              */
/* Sync Jump Width = 2 Tq                         */

/* Ensure the CAN module is in configuration mode.*/

ClCONbits.REQOP = 4
while(ClCONbits.OPMOD != 4);

ClCFGbits.SEG2PHTS = 1; /* Phase seg 2 is freely programmable */
ClCFGbits.SEG2PH = 2; /* Phase seg 2 is 3 Tq. */
ClCFGbits.SEG1PH = 2; /* Phase seg 1 is 3 Tq. */
ClCFGbits.PRSEG = 2; /* Propagation seg 2 is 3 Tq. */
ClCFGbits.SAM = 1; /* Sample bit 3 times. */
ClCFGbits.SJW = 2; /* Sync jump width is 2 Tq */
ClCFGbits.BRP = 3; /* BRP value as calculated in Example 34-11 */

/* CAN bit rate configuration complete. */
```

## 34.11 CAN ERROR MANAGEMENT

### 34.11.1 CAN Bus Errors

The CAN Specification 2.0B defines five different ways of detecting errors:

- Bit Error
- Acknowledge Error
- Form Error
- Stuffing Error
- CRC Error

The bit error and the acknowledge error occur at the bit level; the other three errors occur at the message level.

#### 34.11.1.1 BIT ERROR

A node that is sending a bit on the bus also monitors the bus. A bit error is detected when the bit value that is monitored is different from the bit value that is sent. An exception is when a recessive bit is sent during the stuffed bit stream of the Arbitration field or during the ACK slot. In this case, no bit error occurs when a dominant bit is monitored. A transmitter sending a passive error frame and detecting a dominant bit does not interpret this as a bit error.

#### 34.11.1.2 ACKNOWLEDGE ERROR

In the Acknowledge field of a message, the transmitter checks if the Acknowledge Slot (which it has sent out as a recessive bit) contains a dominant bit. If not, this implies that no other node has received the frame correctly. An acknowledge error has occurred, and as a result, the message must be repeated. No error frame is generated in this case.

#### 34.11.1.3 FORM ERROR

A form error is detected when a fixed-form bit field (EOF, Inter-frame Space, Acknowledge Delimiter or CRC Delimiter) contains one or more illegal bits. For a receiver, a dominant bit during the last bit of EOF is not treated as a form error.

#### 34.11.1.4 STUFFING ERROR

A stuffing error is detected at the bit time of the sixth consecutive equal bit level in a message field that should be coded by the method of bit stuffing.

#### 34.11.1.5 CRC ERROR

The node transmitting a message computes and transmits the CRC corresponding to the transmitted message. Every receiver on the bus performs the same CRC calculation as the transmitter. A CRC error is detected, if the calculated result is not the same as the CRC value obtained from the received message.

## Section 34. Controller Area Network (CAN)

### 34.11.2 Fault Confinement

Every CAN controller on a bus tries to detect the errors outlined above within each message. If an error is found, the discovering node transmits an error frame, thus destroying the bus traffic. The other nodes detect the error caused by the error frame (if they have not already detected the original error) and take appropriate action (that is, discard the current message).

The CAN module maintains two error counters:

- Transmit Error Counter (TERRCNT) - CiTREC<15:8>
- Receive Error Counter (RERRCNT) - CiTREC<7:0>

There are several rules governing how these counters are incremented and/or decremented. That is, a transmitter detecting a fault increments its transmit error counter faster than the listening nodes will increment their receive error counter. This is because there is a good chance that it is the transmitter that is at fault.

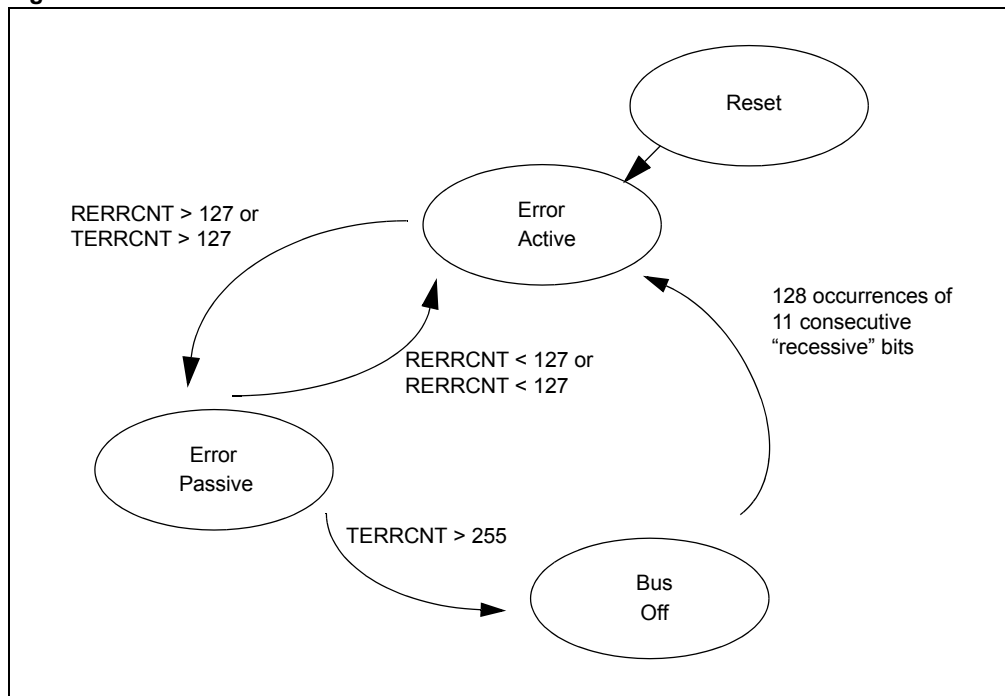
**Note:** The error counters are modified according to the CAN Specification 2.0B.

A node starts out in Error Active mode. When any one of the two Error Counters equals or exceeds a value of 127, the node enters a state known as Error Passive. When the Transmit Error Counter exceeds a value of 255, the node enters the Bus OFF state.

- An Error Active node transmits an Active Error Frame when it detects errors
- An Error Passive node transmits a Passive Error Frame when it detects errors
- A node that is in the Bus OFF state transmits nothing on the bus

In addition, the CAN module employs an error warning feature that warns the user application (when the Transmit Error Counter equals or exceeds 96) before the node enters Error Passive state as illustrated in Figure 34-38.

Figure 34-38: Error Modes



#### 34.11.2.1 TRANSMITTER IN ERROR PASSIVE STATE

The Transmitter Error Passive (TXBP) bit (CiTREC<20>) is set when the Transmit Error Counter equals or exceeds 128 and generates an error interrupt, CERRIF bit (CiINT<12>), upon entry into the Error Passive state. The Transmitter Error Passive flag is cleared automatically by the hardware, if the Transmit Error Counter becomes less than 128.

## 34.11.2.2 RECEIVER IN ERROR PASSIVE STATE

The Receiver Error Passive (RXBP) bit (CiTREC<19>) is set when the Receive Error Counter equals or exceeds 128 and generates an error interrupt, CERRIF bit (CiINT<12>), upon entry into Error Passive state. The Receive Error Passive flag is cleared automatically by the hardware, if the Receive Error Counter becomes less than 128.

## 34.11.2.3 TRANSMITTER IN BUS OFF STATE

The Transmitter Bus OFF (TXBO) bit (CiTREC<21>) is set when the Transmit Error Counter equals or exceeds 256 and generates an error interrupt, CERRIF bit (CiINT<12>).

## 34.11.2.4 TRANSMITTER IN ERROR WARNING STATE

The Transmitter Error Warn (TXWARN) bit (CiTREC<18>) is set when the Transmit Error Counter equals or exceeds 96 and generates an error interrupt, CERRIF bit (CiINT<12>), upon entry into the Error Warn state. The Transmit Error Warn flag is cleared automatically by the hardware, if the Transmit Error Counter becomes less than 96.

## 34.11.2.5 RECEIVER IN ERROR WARNING STATE

The Receiver Error Warn (RXWARN) bit (CiTREC<17>) is set when the Receive Error Counter equals or exceeds 96 and generates an error interrupt, CERRIF bit (CiINT<12>), upon entry into the Error Warn state. The Receive Error Warn flag is cleared automatically by the hardware, if the Receive Error Counter becomes less than 96.

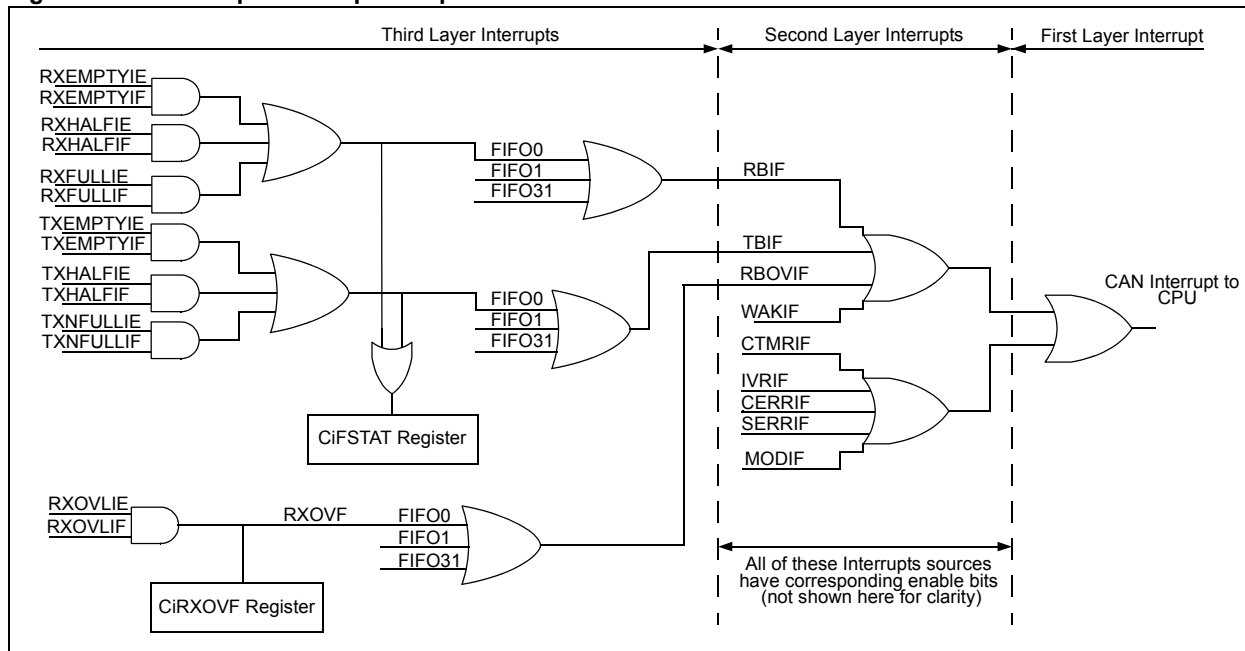
Additionally, there is an Error State Warning Flag (EWARN) bit (CiTREC<16>), which is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWARN is reset if both error counters are less than the error warning limit.



## 34.12 CAN INTERRUPTS

Interrupts in the PIC32 CAN module can be classified into three layers. Lower layer interrupts propagate to higher layers where they are multiplexed into single interrupts. Therefore, the layer 1 interrupt is multiplex of a multiple interrupts in layer 2. Some of the interrupts in layer 2 are a multiplex of interrupts in layer 3 (see Figure 34-39).

**Figure 34-39: Multiple Interrupt Multiplex**



At layer 1, the CAN module generates one interrupt to the CPU called the CANx interrupt. The interrupt is caused by any of the layer 2 interrupts. The CANx Interrupts can be enabled or disabled through the PIC32 Interrupt Controller. For more details on enabling and configuring CPU interrupts on the PIC32 device, refer to the **Section 8. "Interrupts"** (DS61108).

Each of the layer 2 and layer 3 interrupts has an Interrupt Enable bit (IE) bit and Interrupt Status Flag (IF) bit associated with it. The CAN module will set an IF bit when an interrupt event occurs. If the user application has set the IE bit, then the event will be propagated to the next Interrupt layer. User applications can either poll the IF bits or enable interrupts to become aware of the CAN module events.

The ICODE<6:0> bits (CiVEC<6:0>) will contain the code for the latest interrupts. These bits can be used in combination with a jump table for efficient handling of interrupts.

### 34.12.1 Layer 2 Interrupts

The CAN module has nine interrupts at layer 2:

- TBIF – Transmit FIFO Interrupt
- RBIF – Receive FIFO Interrupt
- CTMRIF – CAN Timestamp Timer Rollover Interrupt
- MODIF – CAN Operation Mode Change Interrupt
- RBOVIF – Receive FIFO Overflow Interrupt
- SERRIF – CAN System Error Interrupt
- CERRIF – CAN Bus Error Interrupt
- WAKIF – CAN Wake-up Interrupt
- IVRIF – Invalid CAN Message Interrupt

Of the above interrupts, the Invalid Message Received (IVRIF), the Wake-up (WAKIF), the CAN Timer (CTMRIF) and the CAN Mode Change (MODIF) interrupts have a single source. The FIFO (TBIF/RBIF), FIFO overflow (RBOVIF), CAN Bus Error (CERRIF) and System Error (SERRIF) interrupts have multiple sources.

## 34.12.1.1 INVALID MESSAGE RECEIVED INTERRUPT (IVRIF)

The Invalid Message Received Interrupt (IVRIF) occurs when an error has occurred in the last transmitted message or received message. The specific error that occurred is not available to the user application. To clear the interrupt, the flag bit must be cleared.

## 34.12.1.2 WAKE-UP INTERRUPT (WAKIF)

The Wake-up Interrupt (WAKIF) occurs when CAN bus-activity is detected while the CAN module is in Sleep mode. To clear the interrupt, the flag must be cleared.

## 34.12.1.3 CAN BUS ERROR INTERRUPT (CERRIF)

The CAN Bus Error Interrupt (CERRIF) occurs when there is an error on the CAN bus. The bit is set every time there is change in the current error state of the CAN module with respect to the CAN network. The error states are tracked by the CiTREC register. To clear the interrupt, the flag bit must be cleared. The CERRIF bit will be set once each time the Transmit Error Counter (TERRCNT<7:0>) bits or the Receive Error Counter bits (RERRCNT<7:0>) in the CAN Transmit/Receive Error Counter register (CiTREC) cross a threshold. For example, if the TERRCNT bit (CiTREC<7:0>) goes from 95 to 96, then CERRIF will be set. If this bit is cleared, it will remain clear even if the TERRCNT bit (CiTREC<7:0>) remains above 96. It will be set again if the TERRCNT bit (CiTREC<7:0>) drops below 96 and rises above 95 again, or if the TERRCNT bit (CiTREC<7:0>) goes above 127.

## 34.12.1.4 SYSTEM ERROR INTERRUPT (SERRIF)

A System Error Interrupt (SERRIF) can occur due to two types of system errors, addressing errors and bandwidth errors. Both of these errors are fatal and the CAN module will be stopped. Once the error occurs, the user should wait till the CAN module stops, by polling the CANBUSY bit (CiCON<11>). Once the CAN module has stopped, SERRIF can be cleared by clearing the ON bit (CiCON<15>).

<p><b>Note:</b> A error condition which caused the System Error Interrupt (SERRIF) can only be resolved by turning the CAN module OFF (by clearing the ON bit (CiCON&lt;15&gt;)). Clearing the System Error Interrupt Flag (SERRIF) bit in the CAN Interrupt register (CiINT&lt;12&gt;) will not clear this error condition.</p>
--

## 34.12.1.5 ADDRESSING ERRORS

There are two sources of Addressing errors. Both of these have the same ICODE values (ICODE = 100 0100).

- An addressing error will occur, if a FIFO is configured with an invalid base address. This error will most commonly occur when the FIFO points to an unimplemented address.
- An addressing error will occur if the message destination is illegal, such as attempting to write a received message to program Flash, which is not directly writable

## 34.12.1.6 BUS BANDWIDTH ERROR

A Bus Bandwidth error will occur when the CAN module is unable to write a received CAN message to the location in system memory before the next CAN message arrives. This condition is represented by ICODE = 100 0101. This may occur if some other PIC32 system bus initiator with a higher priority than the CAN module uses the system bus for an extended period of time, thereby preventing the CAN module from storing the received CAN message. The overflow bit in the corresponding FIFO Interrupt register will be set.

## 34.12.1.7 RECEIVE FIFO OVERRUN INTERRUPT (RBOVIF)

The Receive FIFO Overrun Interrupt occurs when the CAN module has received a message but the designated Receive FIFO is full. The CAN module will set the RXOVFLIF flag in the CiFIFOINTn register corresponding to the affected Receive FIFO. This is also reflected in CAN Receive FIFO Overflow Status register (CiRXOVF), if the Receive FIFO Overflow Interrupt Enable (RXOVFLIE) bit is set in the CAN FIFO Interrupt register (CiFIFOINTn<19>).

## Section 34. Controller Area Network (CAN)

The RBOVIF bit in the CAN Interrupt register (CiINT<11>) is the OR reduction of all the bits in the CiRXOVF register. The RBOVIF bit (CiINT<11>) and the RXOVIFn bits in the CAN Receive FIFO Overflow Status register (CiRXOVF<31:0>) are not directly clearable. These need to be cleared by clearing the appropriate RXOVFLIF bit (CiFIFOINTn<3:0>).

### 34.12.1.8 CAN MODE CHANGE INTERRUPT (MODIF)

The CAN Mode Change Interrupt (MODIF) occurs when the OPMOD<2:0> bits (CiCON<23:21>) change indicating a change in the operating mode of the CAN module. The ICODE<6:0> bits (CiVEC<6:0>) will indicate a mode change condition. To clear the interrupt, the flag bit must be cleared.

### 34.12.1.9 CAN TIMESTAMP TIMER INTERRUPT (CTMRIF)

The CAN Timestamp Timer Interrupt (CTMRIF) occurs when CAN Timestamp timer overflows. The ICODE<6:0> bits (CiVEC<6:0>) will indicate a timestamp timer overflow. To clear the interrupt, the flag bit must be cleared.

### 34.12.1.10 CAN TRANSMIT FIFO INTERRUPT (TBIF)

The CAN Transmit FIFO interrupt (TBIF) occurs when there is a change in the status of the Transmit FIFO. This interrupt has multiple layer 3 interrupt sources:

- Transmit FIFO Not Full Interrupt (TXNFULLIF)
- Transmit FIFO Half Full Interrupt (TXHALFIF)
- Transmit FIFO Empty Interrupt (TXEMPTYIF)

The TBIF interrupt cannot be cleared by the application. The interrupt will be cleared when all of the source interrupt conditions terminate.

### 34.12.1.11 CAN RECEIVE FIFO INTERRUPT (RBIF)

The CAN Receive FIFO interrupt (RBIF) occurs when there is change in the status of the Receive FIFO. This interrupt has multiple layer 3 interrupt sources:

- Receive FIFO Full Interrupt (RXFULLIF)
- Receive FIFO Half Full Interrupt (RXHALFIF)
- Receive FIFO Not Empty Interrupt (RXEMPTYIF)

The RBIF interrupt cannot be cleared by the user application. The interrupt will be cleared when all of the source interrupt conditions terminate.

## 34.12.2 Layer 3 Interrupts

The CAN module has the following layer 3 interrupts:

- Transmit FIFO Not Full Interrupt (TXNFULLIF)
- Transmit FIFO Not Half Full Interrupt (TXNHALFIF)
- Transmit FIFO Not Empty Interrupt (TXNEMPTYIF)
- Receive FIFO Full Interrupt (RXFULLIF)
- Receive FIFO Half Full Interrupt (RXHALFIF)
- Receive FIFO Empty Interrupt (RXEMPTYIF)
- Receive FIFO Overflow Interrupt (RXOVLIF)

These interrupts reflect the current status of the Transmit and Receive FIFOs.

## 34.12.3 Interrupt Persistency

The CAN module provides interrupts which indicate the operational status of message FIFOs. These interrupts are persistent, in that they are present till the interrupt causing condition has cleared. The interrupts flags themselves cannot be cleared directly by the user application. For example, the Receive FIFO Not Empty Interrupt (RXNEMPTYIF) will remain set as long as there is at least one message in the Receive FIFO. The following CAN interrupts are persistent:

- Transmit FIFO (TBIF) Interrupt (CiINT<0>)
- Receive FIFO (RBIF) Interrupt (CiINT<1>)
- FIFO Status (FIFOIPx) Interrupt (CiFSTAT<31:0>)
- Transmit FIFO Not Full (TXNFULLIF) Interrupt (CiFIFOINTn<10>)
- Transmit FIFO Half Full (TXHALFIF) Interrupt (CiFIFOINTn<9>)
- Transmit FIFO Empty (TXNEMPTYIF) Interrupt (CiFIFOINTn<8>)
- Receive FIFO Full (RXFULLIF) Interrupt (CiFIFOINTn<10>)
- Receive FIFO Half Full (RXHALFIF) Interrupt (CiFIFOINTn<9>)
- Receive FIFO Not Empty (RXNEMPTYIF) Interrupt (CiFIFOINTn<8>)

Note that the Receive FIFO Overflow (RXOVFLIF) Interrupt (CiFIFOINTn<11>) is sticky and is user-clearable.

## 34.12.4 CAN FIFO Status Register (CiFSTAT)

The CAN FIFO Status register (CiFSTAT) is an indication register. The FIFOIPn bits in the CAN FIFO Status register (CiFSTAT<31:0>) get set when interrupts are enabled in the corresponding CiFIFOINTn registers, and when any of the following interrupt conditions occur:

- Transmit FIFO Not Full (TXNFULLIF) Interrupt (CiFIFOINTn<10>)
- Transmit FIFO Half Full (TXHALFIF) Interrupt (CiFIFOINTn<9>)
- Transmit FIFO Empty (TXNEMPTYIF) Interrupt (CiFIFOINTn<8>)
- Receive FIFO Full (RXFULLIF) Interrupt (CiFIFOINTn<10>)
- Receive FIFO Half Full (RXHALFIF) Interrupt (CiFIFOINTn<9>)
- Receive FIFO Not Empty (RXNEMPTYIF) Interrupt (CiFIFOINTn<8>)

The FIFOIPn bits by themselves are not interrupt sources. The ICODE<6:0> bits (CiVEC<6:0>) will reflect value of the FIFOIPn bits which are set.

## 34.12.5 CAN Receive FIFO Overflow Register (CiRXOVF)

The RXOVFn bits in the CAN Receive FIFO Overflow register (CiRXOVF<31:0>) gets set when a Receive FIFO overflow occurs and when the Receive FIFO Overflow Interrupt Enable (RXOVFLIE) bit (CiFIFOINTn<19>) is set. The RXOVFn bits are source interrupts to the RBOVIF (CiINT<11>) Interrupt. The bits in the CiRXOVF register are not directly clearable, and can be cleared by clearing the RXOVFLIF bit (CiFIFOINTn<3>).

## 34.12.6 Interrupt Code (ICODE) bits

The ICODE<6:0> bits (CiVEC<6:0>) will reflect the highest interrupt that is still pending in the CAN module. The higher the ICODE value, the higher the natural priority of the pending interrupt. The ICODE events are persistent. If an interrupt with a higher natural priority occurs, masking the lower priority in the ICODE register, the lower priority will be shown as soon as the higher priority interrupt is cleared.

For example, consider the following application case:

- FIFO0 causes an interrupt - ICODE<6:0> = 000 0000
- Buffer 5 also has an event that causes an interrupt, causing ICODE to be set to '000 0101'
- The user enters the CAN interrupt handler, services buffer 5 and clears the interrupt
- ICODE<6:0> will now read '000 0000', indicating that buffer 0 still has an interrupt pending, which can then be serviced before returning from the Interrupt Service Routine (ISR)

## 34.12.7 CAN Filter Hit (FILHIT) Bits

The CAN Filter Hit (FILHIT<4:0>) bits in the CAN Interrupt Code register (CiVEC<12:8>) contain the value of the filter that last matched a receiving message. Unlike the ICODE<6:0> bits (CiVEC<6:0>), the FILHIT<4:0> bits (CiVEC<12:8>) have no history, and are only updated when a message is received.

### 34.13 CAN RECEIVED MESSAGE TIME STAMPING

The CAN module can provide a time value of when a message is received. This time stamp will be stored with the received message, if the CAN Receive Message Timer Stamp Time Capture Event Enable (CANCAP) bit in the CAN Control register (CiCON<20>) is set. The CAN module will write a time stamp to the MSGTS bits of the MSGSID field (MSGSID<31:16>) in the Receive Message Buffer. The CAN module obtains the timestamp by capturing the value of CAN Time Stamp Timer (CANTS<15:0>) bits in the CAN Timer register (CiTMR<31:16>) whenever a valid frame has been received. Because the CAN Specification 2.0B defines a frame to be valid, if no errors occurred before the EOF field has been transmitted successfully, the value of CANTS<15:0> bits (CiTMR<31:16>) will be captured right after the EOF.

All received messages will be time stamped, even messages that the CAN module transmits and receives itself. The user application can configure the CAN Time Stamp time interval in terms of system clocks through the CAN Time Stamp Timer Prescaler (CANSTPRE<15:0>) bits in the CAN Timer register (CiTMR<15:0>).

### 34.14 LOW-POWER MODES

#### 34.14.1 Sleep Mode

The user application must ensure that the CAN module is not active when the CPU goes into Sleep mode. To protect the CAN bus system from fatal consequences due to violations of the above rule, the CAN module drives the TXCAN pin into the recessive state while in Sleep mode. The recommended procedure is to bring the CAN module into Disable mode before the CPU is put into Sleep mode.

#### 34.14.2 Idle Mode

When the CPU is in Idle mode, the CAN module powers down if the Stop in Idle Mode (SIDLE) bit in the CAN Control register (CiCON<13>) is '1'. The user application must ensure that the CAN module is not active when the CPU goes into Idle mode and the SIDLE bit (CiCON<13>) is set. To protect the CAN bus system from fatal consequences due to violations of the above rule, the CAN module drives the CiTX pin into the recessive state while the CPU is in Idle mode and the SIDLE bit (CiCON<13>) is set.

#### 34.14.3 Wake-up Functions

The CAN module will monitor the CAN receive (CiRX) pin for activity while the CAN module is in Sleep mode. The CAN module will be sleeping when:

- The device is in Sleep mode
- The device is in Idle mode and the SIDLE bit (CiCON<13>) is set

While in this mode, the CAN module will generate an interrupt, if the Bus Wake-Up Activity Interrupt Enable (WAKIE) bit in the CAN Interrupt register (CiINT<30>) is set and the bus activity is detected.

The CAN module can be programmed to apply a low-pass filter function to the CiRX line while in Disable mode or Sleep/Idle. This feature can be used to protect the CAN module from wake-up due to short glitches on the CAN bus lines. The CAN Bus Line Filter Enable (WAKFIL) bit in the CAN Configuration register (CiCFG<22>) enables or disables the filter while the CAN module is in Sleep mode.

Note that while the CAN module generates the interrupt, the state of the CAN module itself will be not affected by the interrupt.

The following steps describe the recommended procedure for using the wake-up interrupt feature with the PIC32 device Sleep mode.

1. The WAKIE bit CiINT<30> should be set in order to enable the wake-up on bus activity feature.
2. Before placing the device into Sleep mode, switch the CAN operation mode to Disable and wait until the CAN module has entered Disable mode.
3. Put the PIC32 device into Sleep mode.
4. During Sleep, if there is CAN bus activity, the PIC32 device will wake-up from Sleep mode. If the CAN CPU interrupt is enabled, the CPU will execute the CAN ISR.

In the ISR, check whether the CAN Bus Activity Wake-up Interrupt Flag (WAKIF) bit in the CAN Interrupt register (CiINT<14>) is set, and switch the CAN operation mode to Normal. When the switch is complete, the CAN module will be able to receive messages (clear the WAKIF bit (CiINT<14>)). Note that the CAN message that woke the device will be lost.

## 34.15 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the PIC32 device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to Controller Area Network (CAN) module are:

Title	Application Note #
Ethernet Theory of Operation	AN1120

**Note:** Please visit the Microchip web site ([www.microchip.com](http://www.microchip.com)) for additional application notes and code examples for the PIC32 family of devices.

## 34.16 REVISION HISTORY

### Revision A (July 2009)

This is the initial released version of this document.

### Revision B (September 2011)

This revision includes the following updates:

- Examples:
  - Updated CiCFG1 to CiCFG in [Example 34-16](#)
  - Updated [Example 34-17](#)
- Figures:
  - Updated the following labels in [Figure 34-17](#) through [Figure 34-22](#):
    - CFIFOBA is updated to C1FIFOBA
    - CFIFOUA is updated to C1FIFOUA1
    - CFIFOCI is updated to C1FIFOCI1
  - Updated the following labels in [Figure 34-30](#) through [Figure 34-36](#):
    - CFIFOBA is updated to C1FIFOBA
    - CFIFOUA is updated to C1FIFOUA0
    - CFIFOCI is updated to C1FIFOCI1
- Notes:
  - Added a Note above [34.1 “Introduction”](#), which provides information on the complementary documentation
  - Added a note in [34.5.2 “Normal Operation Mode”](#)
- Registers:
  - Updated the register title to **CiCON: CAN Module Control Register** in [Register 34-1](#)
  - Updated [Register 34-11](#) to show correct bit numbers
  - Updated the bit condition of FRESET and UINC bits as S/HC-0 and TXABAT, TXLARB, TXERR bits as R-0 in [Register 34-20](#)
- Sections:
  - Added a new paragraph about checking whether the Transmit FIFO can accommodate the incoming messages, in [34.7.2 “Requesting Transmit of a Message”](#)
  - Updated any references of TXABT to TXABAT in [34.7.4 “Aborting Transmission of a Queued Message”](#)
  - Any references of CiFIFOUA is updated to CiFIFOUA0 in [34.9.3 “Example Receive FIFO Behavior”](#)
- Tables:
  - Updated the EID bits in [Table 34-4](#)
- Updated the following in the entire document:
  - Updated any references of PIC32MX to PIC32
  - Updated any references of system RAM to device RAM
  - Updated any references of TXPRI to TXPR
  - Updated any references of CMSG0SID to CMSGSID
- Additional minor corrections such as text and formatting updates were incorporated throughout the document



---

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

---

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

**Trademarks**

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC<sup>32</sup> logo, rPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, REAL ICE, rLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2009-2011, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-61341-645-7

**QUALITY MANAGEMENT SYSTEM**  
**CERTIFIED BY DNV**  
**== ISO/TS 16949:2009 ==**

*Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC<sup>®</sup> MCUs and dsPIC<sup>®</sup> DSCs, KEELOQ<sup>®</sup> code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*



# MICROCHIP

## Worldwide Sales and Service

### AMERICAS

**Corporate Office**  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 480-792-7200  
Fax: 480-792-7277  
Technical Support:  
<http://www.microchip.com/support>  
Web Address:  
[www.microchip.com](http://www.microchip.com)

**Atlanta**  
Duluth, GA  
Tel: 678-957-9614  
Fax: 678-957-1455

**Boston**  
Westborough, MA  
Tel: 774-760-0087  
Fax: 774-760-0088

**Chicago**  
Itasca, IL  
Tel: 630-285-0071  
Fax: 630-285-0075

**Cleveland**  
Independence, OH  
Tel: 216-447-0464  
Fax: 216-447-0643

**Dallas**  
Addison, TX  
Tel: 972-818-7423  
Fax: 972-818-2924

**Detroit**  
Farmington Hills, MI  
Tel: 248-538-2250  
Fax: 248-538-2260

**Indianapolis**  
Noblesville, IN  
Tel: 317-773-8323  
Fax: 317-773-5453

**Los Angeles**  
Mission Viejo, CA  
Tel: 949-462-9523  
Fax: 949-462-9608

**Santa Clara**  
Santa Clara, CA  
Tel: 408-961-6444  
Fax: 408-961-6445

**Toronto**  
Mississauga, Ontario,  
Canada  
Tel: 905-673-0699  
Fax: 905-673-6509

### ASIA/PACIFIC

**Asia Pacific Office**  
Suites 3707-14, 37th Floor  
Tower 6, The Gateway  
Harbour City, Kowloon  
Hong Kong  
Tel: 852-2401-1200  
Fax: 852-2401-3431

**Australia - Sydney**  
Tel: 61-2-9868-6733  
Fax: 61-2-9868-6755

**China - Beijing**  
Tel: 86-10-8569-7000  
Fax: 86-10-8528-2104

**China - Chengdu**  
Tel: 86-28-8665-5511  
Fax: 86-28-8665-7889

**China - Chongqing**  
Tel: 86-23-8980-9588  
Fax: 86-23-8980-9500

**China - Hangzhou**  
Tel: 86-571-2819-3187  
Fax: 86-571-2819-3189

**China - Hong Kong SAR**  
Tel: 852-2401-1200  
Fax: 852-2401-3431

**China - Nanjing**  
Tel: 86-25-8473-2460  
Fax: 86-25-8473-2470

**China - Qingdao**  
Tel: 86-532-8502-7355  
Fax: 86-532-8502-7205

**China - Shanghai**  
Tel: 86-21-5407-5533  
Fax: 86-21-5407-5066

**China - Shenyang**  
Tel: 86-24-2334-2829  
Fax: 86-24-2334-2393

**China - Shenzhen**  
Tel: 86-755-8203-2660  
Fax: 86-755-8203-1760

**China - Wuhan**  
Tel: 86-27-5980-5300  
Fax: 86-27-5980-5118

**China - Xian**  
Tel: 86-29-8833-7252  
Fax: 86-29-8833-7256

**China - Xiamen**  
Tel: 86-592-2388138  
Fax: 86-592-2388130

**China - Zhuhai**  
Tel: 86-756-3210040  
Fax: 86-756-3210049

### ASIA/PACIFIC

**India - Bangalore**  
Tel: 91-80-3090-4444  
Fax: 91-80-3090-4123

**India - New Delhi**  
Tel: 91-11-4160-8631  
Fax: 91-11-4160-8632

**India - Pune**  
Tel: 91-20-2566-1512  
Fax: 91-20-2566-1513

**Japan - Yokohama**  
Tel: 81-45-471- 6166  
Fax: 81-45-471-6122

**Korea - Daegu**  
Tel: 82-53-744-4301  
Fax: 82-53-744-4302

**Korea - Seoul**  
Tel: 82-2-554-7200  
Fax: 82-2-558-5932 or  
82-2-558-5934

**Malaysia - Kuala Lumpur**  
Tel: 60-3-6201-9857  
Fax: 60-3-6201-9859

**Malaysia - Penang**  
Tel: 60-4-227-8870  
Fax: 60-4-227-4068

**Philippines - Manila**  
Tel: 63-2-634-9065  
Fax: 63-2-634-9069

**Singapore**  
Tel: 65-6334-8870  
Fax: 65-6334-8850

**Taiwan - Hsin Chu**  
Tel: 886-3-5778-366  
Fax: 886-3-5770-955

**Taiwan - Kaohsiung**  
Tel: 886-7-536-4818  
Fax: 886-7-330-9305

**Taiwan - Taipei**  
Tel: 886-2-2500-6610  
Fax: 886-2-2508-0102

**Thailand - Bangkok**  
Tel: 66-2-694-1351  
Fax: 66-2-694-1350

### EUROPE

**Austria - Wels**  
Tel: 43-7242-2244-39  
Fax: 43-7242-2244-393

**Denmark - Copenhagen**  
Tel: 45-4450-2828  
Fax: 45-4485-2829

**France - Paris**  
Tel: 33-1-69-53-63-20  
Fax: 33-1-69-30-90-79

**Germany - Munich**  
Tel: 49-89-627-144-0  
Fax: 49-89-627-144-44

**Italy - Milan**  
Tel: 39-0331-742611  
Fax: 39-0331-466781

**Netherlands - Drunen**  
Tel: 31-416-690399  
Fax: 31-416-690340

**Spain - Madrid**  
Tel: 34-91-708-08-90  
Fax: 34-91-708-08-91

**UK - Wokingham**  
Tel: 44-118-921-5869  
Fax: 44-118-921-5820

08/02/11