Section 21. Enhanced Controller Area Network (ECAN™)

HIGHLIGHTS

This section of the manual contains the following major topics:

21.1   INTRODUCTION

The PIC24H Enhanced Controller Area Network (ECAN™) module implements the CAN Protocol 2.0B, used primarily in industrial and automotive applications. This asynchronous serial data communication protocol provides reliable communications in electrically noisy environments. The PIC24H device family integrates up to two ECAN modules. Figure 21-1 illustrates a typical CAN bus topology.

Figure 21-1:   Typical CAN Bus Network
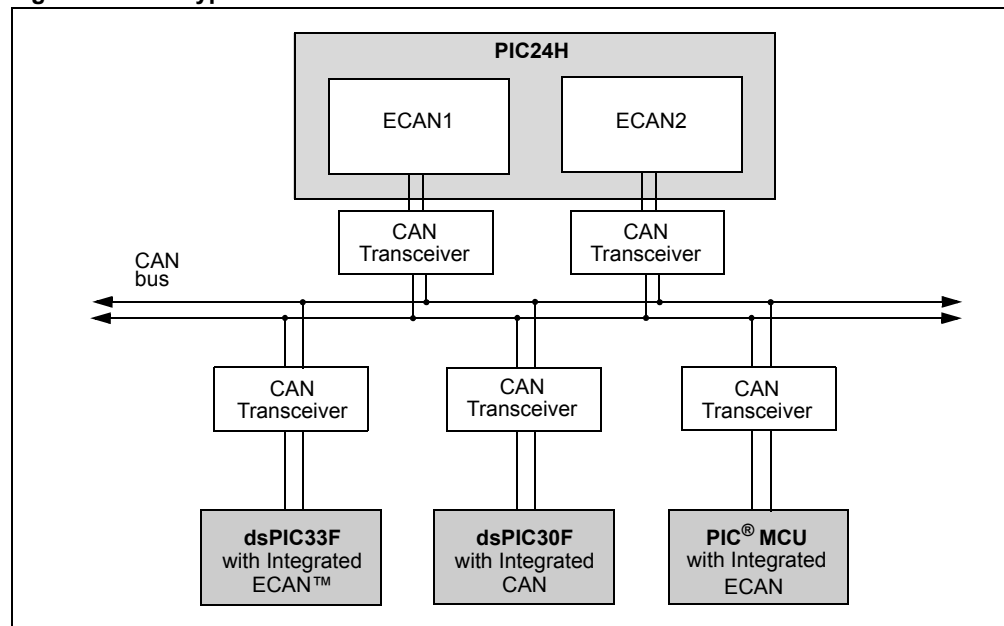
The ECAN module supports the following key features:

Standards Compliance:

- Full CAN 2.0B compliance
- Programmable bit rate up to 1 Mbps

Message Reception:

- 32 message buffers – all of them can be used for reception
- 16 acceptance filters for message filtering
- Three acceptance filter mask registers for message filtering
- Automatic response to Remote Transmit Request
- Up to 32-message deep First In First Out (FIFO) buffer
- DeviceNet™ addressing support
- DMA interface for message reception

Message Transmission:

- Eight message buffers configurable for message transmission
- User-defined priority levels for message buffers used for transmission
- DMA interface for message transmission

Others:

- Loopback, Listen All Messages and Listen Only modes for self-test, system diagnostics, and bus monitoring
- Low-power operating modes

Section 21. Enhanced Controller Area Network (ECAN™)

Figure 21-2 illustrates the general structure of the ECAN module and its interaction with the DMA Controller and DMA RAM.

Figure 21-2: ECAN™ Interaction with DMA
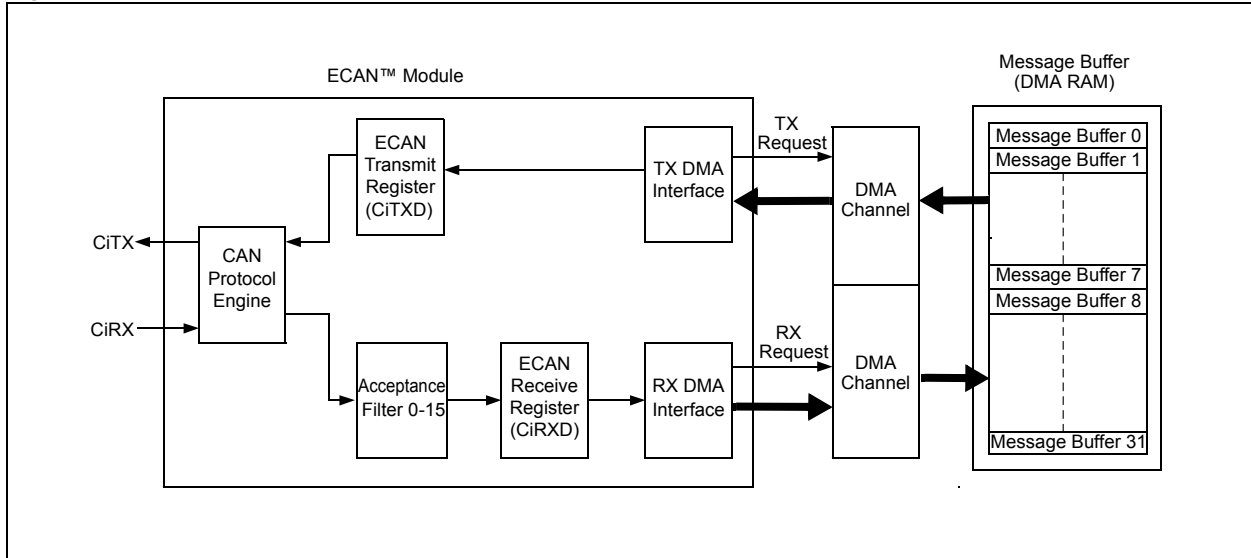
21.1.1 ECAN Module

The ECAN module consists of a protocol engine, message acceptance filters, and separate transmit and receive DMA interfaces. The protocol engine transmits and receives messages to and from the CAN bus (as per CAN bus 2.0B protocol). The user-configurable acceptance filters are used by the module to examine the received message to determine if it should be stored in the DMA message buffer or discarded.

For received messages, the receive DMA interface generates a receive data interrupt to initiate a DMA cycle. The receive DMA channel reads data from the CiRXD register and writes it into the message buffer.

For transmit messages, the transmit DMA interface generates a transmit data interrupt to start a DMA cycle. The transmit DMA channel reads from the message buffer and writes to the CiTXD register for message transmission.

21.1.2 Message Buffers

The ECAN module supports up to 32 message buffers for storing data transmitted or received on the CAN bus. These buffers are located in DMA RAM. Message buffers 0-7 can be configured for either transmit or receive operation. Message buffers 8-31 are receive-only buffers and cannot be used for Message Transmission.

21.1.3 DMA Controller

The DMA controller acts as an interface between the message buffers and ECAN to transfer data back and forth without CPU intervention. The DMA controller supports up to eight channels for transferring data between DMA RAM and the PIC24H peripherals. Two separate DMA channels are needed to support CAN message transmission and CAN message reception.

Each DMA channel has a DMA Request register (DMAxREQ), which is used by the user application to assign an interrupt event to trigger a DMA-based message transfer.

21.2 CAN MESSAGE FORMATS

The CAN bus protocol uses asynchronous communication. Information is passed from transmitters to receivers in data frames, which are composed of byte fields that define the contents of the data frame, as shown in Figure 21-3.

Each frame begins with a Start-of-Frame (SOF) bit and terminates with an End-of-Frame (EOF) bit field. The Start-of-Frame is followed by 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 variable length, ranging from 0 to 8 bytes. Error protection is provided by the Cyclic Redundancy Check (CRC) and Acknowledgement (ACK) fields.

Figure 21-3: CAN Bus Message Frame

SOF	ARBITRATION	CONTROL	DATA	CRC	ACK	EOF
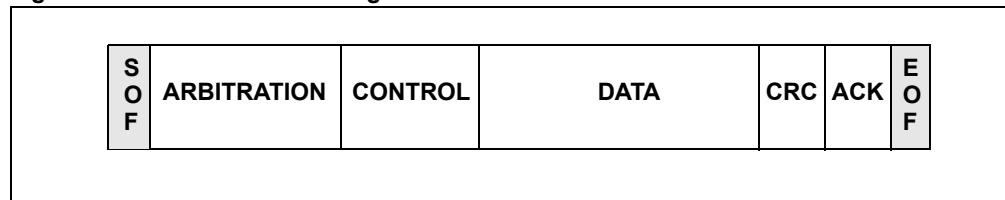
The CAN bus protocol supports five frame types:

- **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 2.0B specification 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 versions of CAN Bus specifications:

- **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 identifier

The PIC24H ECAN module is compliant with the CAN 2.0B active specification, while providing enhanced message filtering capabilities.

> **Note:** For detailed information on the CAN protocol, refer to the Bosch CAN bus specification.

21.2.1   Standard Data Frame

The standard data frame message begins with a Start-of-Frame bit followed by a 12-bit Arbitration field, as shown in Figure 21-4. The Arbitration field contains an 11-bit identifier and the Remote Transmit Request (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 4 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 variable length, ranging from 0 to 8 bytes. The number of bytes is user-selectable.

The Data field is followed by the Cyclic Redundancy Check (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 irrespective of the result of the acceptance filter comparison.

The last field is the End-of-Frame (EOF) field, which consists of 7 recessive bits that indicate the end of the message.

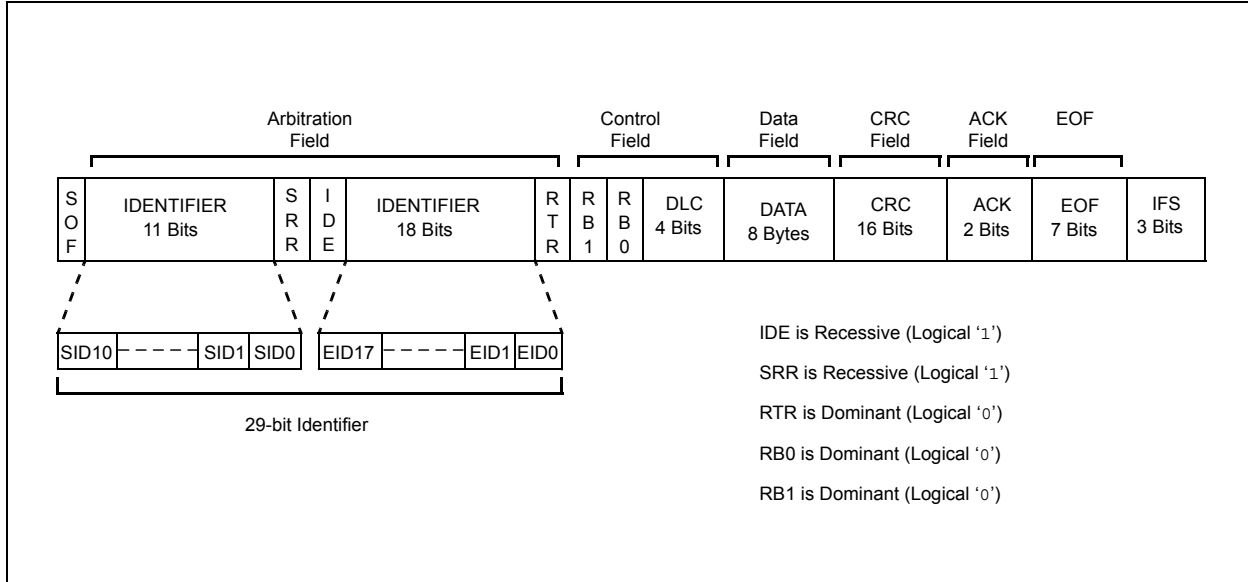**Figure 21-4:   Format of the Standard Data Frame**
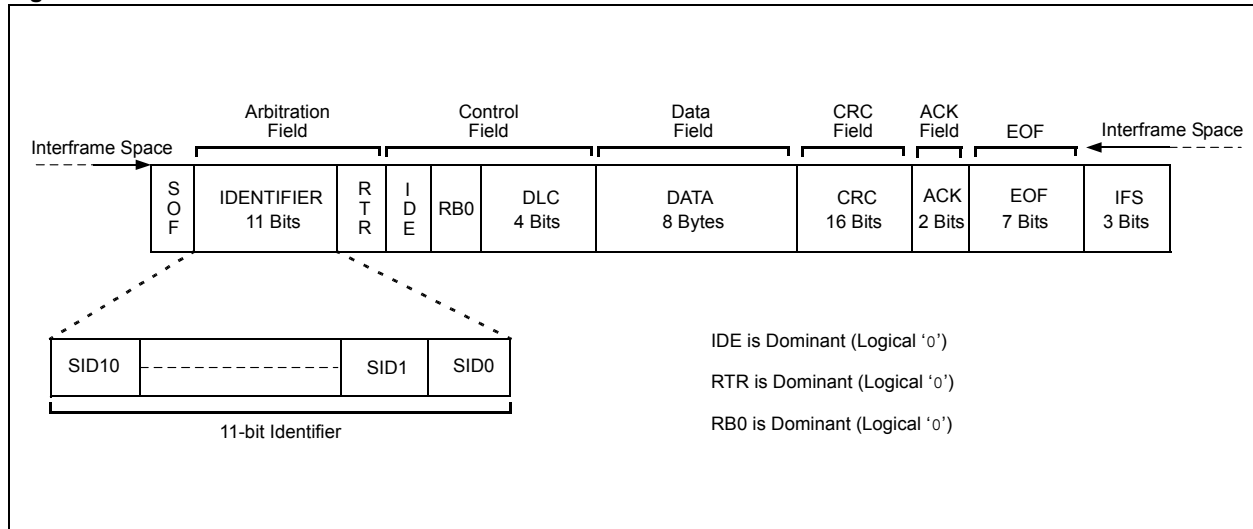
21.2.2 Extended Data Frame

An extended data frame begins with an SOF bit followed by a 31-bit Arbitration field, as shown in Figure 21-5. 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. 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 7 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 4 bits in the Control field are the Data Length Code, 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.

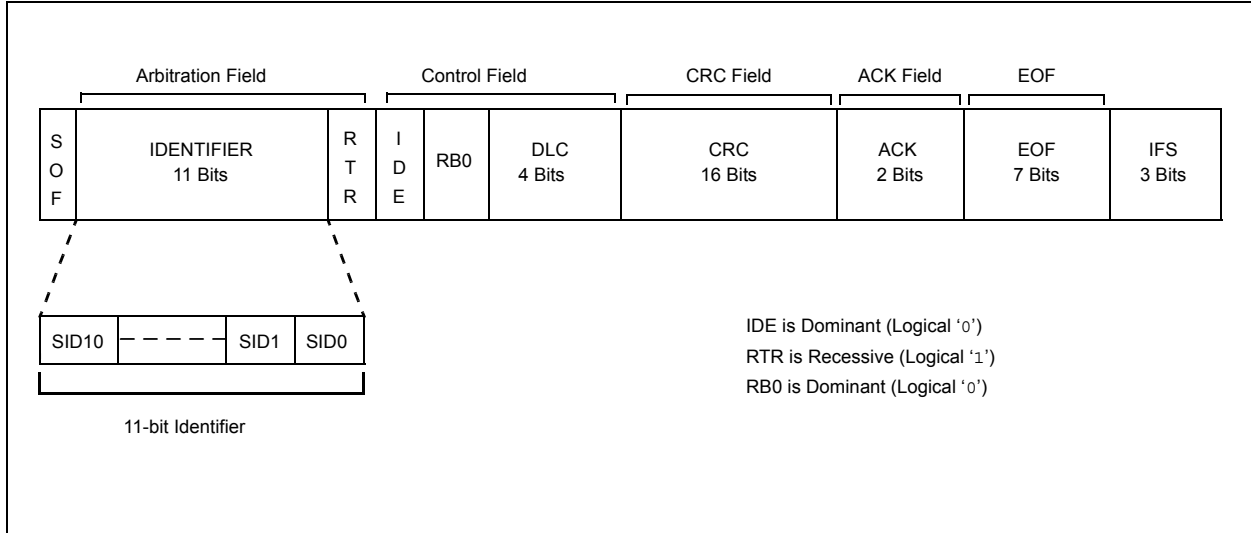**Figure 21-5:   Format of the Extended Data Frame**

21.2.3 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 (see Figure 21-6) or extended format (see Figure 21-7).
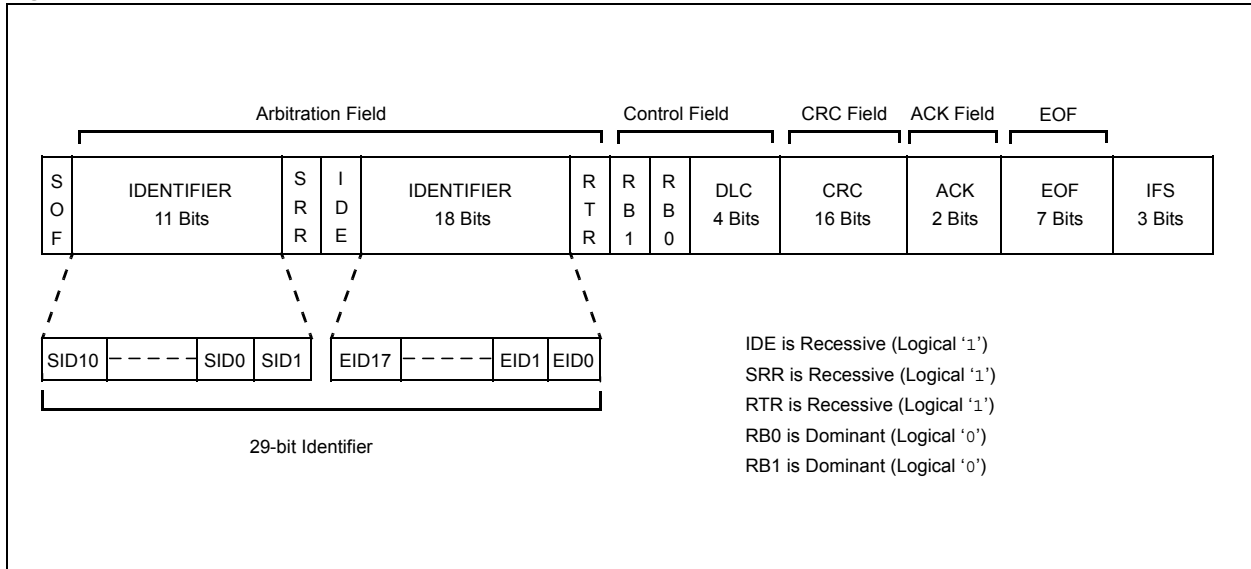
A Remote frame is similar to a data frame, with the following exceptions:

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

Figure 21-6: Format of the Standard Remote Frame

IDE is Dominant (Logical '0')
RTR is Recessive (Logical '1')
RB0 is Dominant (Logical '0')

Figure 21-7: Format of the Extended Remote Frame

IDE is Recessive (Logical '1')
SRR is Recessive (Logical '1')
RTR is Recessive (Logical '1')
RB0 is Dominant (Logical '0')
RB1 is Dominant (Logical '0')

21.2.4 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 8 recessive bits and allows the bus nodes to restart communications 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 6 consecutive dominant bits, which forces all other nodes on the network to generate Error Echo Flags, thereby resulting in a series of 6 to 12 dominant bits on the bus.
- **Error Passive Flag** – contains 6 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 communications of any other node on the network.

21.2.5 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 yet 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 Flag field with 6 dominant bits followed by an Overload Delimiter field with 8 recessive bits. A node can generate a maximum of two sequential Overload Frames to delay the start of the next message.

21.2.6 Interframe Space

Interframe Space separates successive frames being transmitted on the CAN bus. It consists of at least 3 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 8 recessive bits are 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.

21.3   ECAN REGISTERS

The ECAN module has a large number of Special Function Registers (SFRs) that are used to configure message acceptance filters and message buffers. To enable effective use of data RAM space, multiple sets of SFRs are mapped onto the same set of memory addresses. The SFR Map Window Select (WIN) bit in ECAN Control Register 1 (CiCTRL1<0>) is used to selectively access one of these sets of SFRs.

If CiCTRL1<WIN> = 1, the message acceptance filters, masks and filter buffer pointer registers are accessed by the user application.

If CiCTRL1<WIN> = 0, the buffer control and status registers and the transmit and receive data registers are accessed by the user application.

21.3.1   ECAN Baud Rate Control Registers

• **CiCFG1: ECAN Baud Rate Configuration Register 1**

This register contains control bits to set the period of each time quantum, using the baud rate prescaler, and specifies synchronization jump width in terms of time quanta (see Register 21-1).

> **Note:**   The 'i' shown in register identifiers refers to ECAN 1 or ECAN 2.

• **CiCFG2: ECAN Baud Rate Configuration Register 2**

This register is used to program the number of time quanta in each CAN bit segment, including the propagation and phase segments 1 and 2 (see Register 21-2).

21.3.2   ECAN Message Filter Registers

• **CiFEN1: ECAN Acceptance Filter Enable Register**

This register enables/disables acceptance filters 0-15 for message filtering (see Register 21-3).

• **CiRXFnSID: ECAN Acceptance Filter Standard Identifier Register n (n = 0-15)**

These 16 registers specify the standard message identifier for acceptance filters 0-15. The identifier bits are selectively masked against the incoming message identifier to determine if the message should be accepted or rejected (see Register 21-4). These registers are only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1 = use filter window).

• **CiRXFnEID: ECAN Acceptance Filter Extended Identifier Register n (n = 0-15)**

These 16 registers specify the extended message identifier for acceptance filters 0-15. The identifier bits are selectively masked against the incoming message identifier to determine if the message should be accepted or rejected (see Register 21-5). These registers are only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

• **CiRXMnSID: ECAN Acceptance Filter Mask Standard Identifier Register n (n = 0-2)**

These three registers specify the standard identifier mask bits for Acceptance Masks 0, 1 and 2. Any acceptance filter can optionally select one of these mask registers to selectively compare the identifier bits (see Register 21-6). These registers are only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

• **CiRXMnEID: ECAN Acceptance Filter Mask Extended Identifier Register n (n = 0-2)**

There are three pairs of registers that specify Acceptance Mask bits for Mask 0, 1 and 2. Any acceptance filter can optionally select one of the mask registers to selectively compare the identifier bits (see Register 21-7). These registers are only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

- **CiFMSKSEL1: ECAN Filter 7-0 Mask Selection Register 1**

    This register is used with CiFMSKSEL2 to select the Acceptance mask for acceptance filter 0-7 (see Register 21-8).

- **CiFMSKSEL2: ECAN Filter 15-8 Mask Selection Register 2**

    This register is used with CiFMSKSEL1 to select the Acceptance mask for acceptance filter 8-15 (see Register 21-9).

- **CiBUFPNT1: ECAN Filter 0-3 Buffer Pointer Register 1**

    This register specifies the message buffer to use for storing messages accepted by acceptance filters 0-3 (see Register 21-10). This register is only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

- **CiBUFPNT2: ECAN Filter 4-7 Buffer Pointer Register 2**

    This register specifies the message buffer to use for storing messages accepted by acceptance filters 4-7 (see Register 21-11). This register is only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

- **CiBUFPNT3: ECAN Filter 8-11 Buffer Pointer Register 3**

    This register specifies the message buffer to use for storing messages accepted by acceptance filters 8-11 (see Register 21-12). This register is only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

- **CiBUFPNT4: ECAN Filter 12-15 Buffer Pointer Register 4**

    This register specifies the message buffer to use for storing messages accepted by acceptance filters 12-15 (see Register 21-13). This register is only accessible by the user application when the WIN bit is set (CiCTRL1<0> = 1).

21.3.3   ECAN Message Buffer Status Registers

- **CiRXFUL1: ECAN Receive Buffer Full Register 1**

    Paired with CiRXFUL2, this register indicates buffer full status for Message buffers 0-31. When a received message is stored into a message buffer, the respective buffer full flag is set (see Register 21-14). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0 = use buffer window).

- **CiRXFUL2: ECAN Receive Buffer Full Register 2**

    Paired with CiRXFUL1, this register indicates buffer full status for Message buffers 0-31. When a received message is stored into a message buffer, the respective buffer full flag is set (see Register 21-15). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiRXOVF1: ECAN Receive Buffer Overflow Register 1**

    Paired with CiRXOVF2, this register indicates overflow status for Message buffers 0-31. When a received message is stored into a message buffer and the respective buffer full flag is set, the message is lost, and the respective buffer overflow flag is set (see Register 21-16). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiRXOVF2: ECAN Receive Buffer Overflow Register 2**

    Paired with CiRXOVF1, this register indicates overflow status for Message buffers 0-31. When a received message is stored into a message buffer and the respective buffer full flag is set, the message is lost, and the respective buffer overflow flag is set (see Register 21-17). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

21.3.4   ECAN FIFO Control/Status Registers

- **CiFCTRL: ECAN FIFO Control Register**

This register controls operation of the receive buffer FIFO. It specifies the FIFO start address and number of message buffers reserved for ECAN in DMA RAM (see Register 21-18). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiFIFO: ECAN FIFO Status Register**

This register contains write and read pointers. The write pointer indicates which buffer contains the most-recently received data. The read pointer tells the user application which buffer to read next (see Register 21-19). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

21.3.5   ECAN Interrupt Control/Status Registers

- **CiINTF: ECAN Interrupt Flag Register**

This register provides the status of various interrupt sources in the ECAN module (see Register 21-20). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiINTE: ECAN Interrupt Enable Register**

This register selectively enables/disables 7 main sources of interrupt: transmit buffer interrupt, receive buffer interrupt, receive buffer overflow interrupt, FIFO almost full interrupt, error interrupt, wake-up interrupt and invalid message received interrupt (see Register 21-21). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiVEC: ECAN Interrupt Code Register**

This register provides interrupt code bits that can be used with a jump table for efficient handling of interrupts (see Register 21-22). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

21.3.6   ECAN Control and Error Counter Registers

- **CiCTRL1: ECAN Control Register 1**

This register sets the ECAN module operation modes (see Register 21-23).

- **CiCTRL2: ECAN Control Register 2**

This register contains the DeviceNet™ filtering control bits (see Register 21-24).

- **CiTRmnCON: ECAN TX/RX Buffer m Control Register (m = 0,2,4,6; n = 1,3,5,7)**

These registers configure and control message buffers (see Register 21-25). This register is only accessible by the user application when the WIN bit is cleared (CiCTRL1<0> = 0).

- **CiEC: ECAN Transmit/Receive Error Count Register**

This register counts transmit and receive errors. The user application can read this register to determine the current number of transmit and receive errors (see Register 21-26).

- **CiRXD: ECAN Receive Data Register**

This register temporarily holds every received word. This is the register from which the DMA controller reads data into the DMA buffer.

- **CiTXD: ECAN Transmit Data Register**

This register temporarily holds every transmission. This is the register to which the DMA Controller writes data from the DMA buffer.

Register 21-1:   CiCFG1: ECAN Baud Rate Configuration Register 1

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 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
SJW<1:0>		BRP<5:0>					
bit 7							bit 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 15-8   **Unimplemented:** Read as '0'

bit 7-6   **SJW<1:0>:** Synchronization Jump Width bits

11 = Length is 4 x TQ
10 = Length is 3 x TQ
01 = Length is 2 x TQ
00 = Length is 1 x TQ

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

11 1111 = TQ = 2 x 64 x 1/F_{CAN}

•

•

•

00 0010 = TQ = 2 x 3 x 1/F_{CAN}
00 0001 = TQ = 2 x 2 x 1/F_{CAN}
00 0000 = TQ = 2 x 1 x 1/F_{CAN}

Register 21-2: CiCFG2: ECAN Baud Rate Configuration Register 2

U-0	R/W-x	U-0	U-0	U-0	R/W-x	R/W-x	R/W-x
—	WAKFIL	—	—	—		SEG2PH<2:0>	
bit 15							bit 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
SEG2PHTS	SAM		SEG1PH<2:0>			PRSEG<2:0>	
bit 7							bit 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 15   **Unimplemented:** Read as '0'

bit 14   **WAKFIL:** Select CAN Bus Line Filter for Wake-up bit

1 = Use CAN bus line filter for wake-up
0 = CAN bus line filter is not used for wake-up

bit 13-11   **Unimplemented:** Read as '0'

bit 10-8   **SEG2PH<2:0>:** Phase Segment 2 bits

111 = Length is 8 x TQ

•

•

•

000 = Length is 1 x TQ

bit 7   **SEG2PHTS:** Phase Segment 2 Time Select bit

1 = Freely programmable
0 = Maximum of SEG1PH bits or Information Processing Time (IPT), whichever is greater

bit 6   **SAM:** Sample CAN Bus Line bit

1 = Bus line is sampled three times at the sample point
0 = Bus line is sampled once at the sample point

bit 5-3   **SEG1PH<2:0>:** Phase Segment 1 bits

111 = Length is 8 x TQ

•

•

•

000 = Length is 1 x TQ

bit 2-0   **PRSEG<2:0>:** Propagation Time Segment bits

111 = Length is 8 x TQ

•

•

•

000 = Length is 1 x TQ

Register 21-3: CiFEN1: ECAN Acceptance Filter Enable Register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
FLTEN15	FLTEN14	FLTEN13	FLTEN12	FLTEN11	FLTEN10	FLTEN9	FLTEN8
bit 15							bit 8

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
FLTEN7	FLTEN6	FLTEN5	FLTEN4	FLTEN3	FLTEN2	FLTEN1	FLTEN0
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0 **FLTENn:** Enable Filter n bits

1 = Enable filter n to accept messages
0 = Disable filter n

Register 21-4: CiRXFnSID: ECAN Acceptance Filter Standard Identifier Register n (n = 0-15)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 15							bit 8

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDE	—	EID17	EID16
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-5 **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 4 **Unimplemented:** Read as '0'

bit 3 **EXIDE:** Extended Identifier Enable bit

<u>If MIDE = 1:</u>

1 = Match only messages with extended identifier addresses
0 = Match only messages with standard identifier addresses

<u>If MIDE = 0:</u>
Ignore EXIDE bit.

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID<17:16>:** 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:	If no mask is applied to a filter, the filter will only accept Standard frames. The filter will not accept extended frames even if the EXIDE bit is set to '1'.

Register 21-5:   CiRXFnEID: ECAN Acceptance Filter Extended Identifier Register n (n = 0-15)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 15							bit 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
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0   **EID<15:0>:** 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

Register 21-6:   CiRXMnSID: ECAN Acceptance Filter Mask Standard Identifier Register n (n = 0-2)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 15							bit 8

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	MIDE	—	EID17	EID16
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-5   **SID<10:0>:** Standard Identifier bits

   1 = Include bit SIDx in filter comparison
   0 = Bit SIDx is "don't care" in filter comparison

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

bit 3   **MIDE:** Identifier Receive Mode bit

   1 = Match only message types (standard or extended address) that correspond to EXIDE bit in filter
   0 = Match either standard or extended address message if filters match
     (i.e., if (Filter SID) = (Message SID) or if (Filter SID/EID) = (Message SID/EID))

bit 2   **Unimplemented:** Read as '0'

bit 1-0   **EID<17:16>:** Extended Identifier bits

   1 = Include bit EIDx in filter comparison
   0 = Bit EIDx is a "don't care" in filter comparison

Register 21-7: CiRXMnEID: ECAN Acceptance Filter Mask Extended Identifier Register n (n = 0-2)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 15							bit 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
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-0 **EID<15:0>:** Extended Identifier bits

1 = Include bit EIDx in filter comparison
0 = Bit EIDx is "don't care" in filter comparison

Register 21-8: CiFMSKSEL1: ECAN Filter 7-0 Mask Selection Register 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F7MSK<1:0>		F6MSK<1:0>		F5MSK<1:0>		F4MSK<1:0>	
bit 15							bit 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
F3MSK<1:0>		F2MSK<1:0>		F1MSK<1:0>		F0MSK<1:0>	
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-14 **F7MSK<1:0>:** Mask Source for Filter 7 bits

11 = Reserved
10 = Acceptance Mask 2 registers contain mask
01 = Acceptance Mask 1 registers contain mask
00 = Acceptance Mask 0 registers contain mask

bit 13-12 **F6MSK<1:0>:** Mask Source for Filter 6 bits (same values as bits 15-14)
bit 11-10 **F5MSK<1:0>:** Mask Source for Filter 5 bits (same values as bits 15-14)
bit 9-8 **F4MSK<1:0>:** Mask Source for Filter 4 bits (same values as bits 15-14)
bit 7-6 **F3MSK<1:0>:** Mask Source for Filter 3 bits (same values as bits 15-14)
bit 5-4 **F2MSK<1:0>:** Mask Source for Filter 2 bits (same values as bits 15-14)
bit 3-2 **F1MSK<1:0>:** Mask Source for Filter 1 bits (same values as bits 15-14)
bit 1-0 **F0MSK<1:0>:** Mask Source for Filter 0 bits (same values as bits 15-14)

Register 21-9:   CiFMSKSEL2: ECAN Filter 15-8 Mask Selection Register 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F15MSK<1:0>		F14MSK<1:0>		F13MSK<1:0>		F12MSK<1:0>	
bit 15							bit 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
F11MSK<1:0>		F10MSK<1:0>		F9MSK<1:0>		F8MSK<1:0>	
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-14   **F15MSK<1:0>:** Mask Source for Filter 15 bits
　　　　　　　11 = Reserved
　　　　　　　10 = Acceptance Mask 2 registers contain mask
　　　　　　　01 = Acceptance Mask 1 registers contain mask
　　　　　　　00 = Acceptance Mask 0 registers contain mask

bit 13-12   **F14MSK<1:0>:** Mask Source for Filter 14 bits (same values as bits 15-14)

bit 11-10   **F13MSK<1:0>:** Mask Source for Filter 13 bits (same values as bits 15-14)

bit 9-8   **F12MSK<1:0>:** Mask Source for Filter 12 bits (same values as bits 15-14)

bit 7-6   **F11MSK<1:0>:** Mask Source for Filter 11 bits (same values as bits 15-14)

bit 5-4   **F10MSK<1:0>:** Mask Source for Filter 10 bits (same values as bits 15-14)

bit 3-2   **F9MSK<1:0>:** Mask Source for Filter 9 bits (same values as bits 15-14)

bit 1-0   **F8MSK<1:0>:** Mask Source for Filter 8 bits (same values as bits 15-14)

Register 21-10:   CiBUFPNT1: ECAN Filter 0-3 Buffer Pointer Register 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F3BP<3:0>				F2BP<3:0>			
bit 15							bit 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
F1BP<3:0>				F0BP<3:0>			
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-12   **F3BP<3:0>:** RX Buffer mask for Filter 3 bits
　　　　　　　1111 = Filter hits received in RX FIFO buffer
　　　　　　　1110 = Filter hits received in RX Buffer 14
　　　　　　　•
　　　　　　　•
　　　　　　　•
　　　　　　　0001 = Filter hits received in RX Buffer 1
　　　　　　　0000 = Filter hits received in RX Buffer 0

bit 11-8   **F2BP<3:0>:** RX Buffer mask for Filter 2 bits (same values as bits 15-12)

bit 7-4   **F1BP<3:0>:** RX Buffer mask for Filter 1 bits (same values as bits 15-12)

bit 3-0   **F0BP<3:0>:** RX Buffer mask for Filter 0 bits (same values as bits 15-12)

Register 21-11: CiBUFPNT2: ECAN Filter 4-7 Buffer Pointer Register 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F7BP<3:0>				F6BP<3:0>			
bit 15							bit 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
F5BP<3:0>				F4BP<3:0>			
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-12   **F7BP<3:0>:** RX Buffer mask for Filter 7 bits
1111 = Filter hits received in RX FIFO buffer
1110 = Filter hits received in RX Buffer 14
•
•
•
0001 = Filter hits received in RX Buffer 1
0000 = Filter hits received in RX Buffer 0

bit 11-8   **F6BP<3:0>:** RX Buffer mask for Filter 6 bits (same values as bits 15-12)

bit 7-4   **F5BP<3:0>:** RX Buffer mask for Filter 5 bits (same values as bits 15-12)

bit 3-0   **F4BP<3:0>:** RX Buffer mask for Filter 4 bits (same values as bits 15-12)

Register 21-12: CiBUFPNT3: ECAN Filter 8-11 Buffer Pointer Register 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
F11BP<3:0>				F10BP<3:0>			
bit 15							bit 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
F9BP<3:0>				F8BP<3:0>			
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-12   **F11BP<3:0>:** RX Buffer mask for Filter 11
1111 = Filter hits received in RX FIFO buffer
1110 = Filter hits received in RX Buffer 14
•
•
•
0001 = Filter hits received in RX Buffer 1
0000 = Filter hits received in RX Buffer 0

bit 11-8   **F10BP<3:0>:** RX Buffer mask for Filter 10 (same values as bit 15-12)

bit 7-4   **F9BP<3:0>:** RX Buffer mask for Filter 9 (same values as bit 15-12)

bit 3-0   **F8BP<3:0>:** RX Buffer mask for Filter 8 (same values as bit 15-12)

Register 21-13: CiBUFPNT4: ECAN Filter 12-15 Buffer Pointer Register 4

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	F15BP	<3:0>		F14BP	<3:0>		
bit 15							bit 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
F13BP	<3:0>			F12BP	<3:0>		
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-12 **F15BP<3:0>:** RX Buffer mask for Filter 15 bits
1111 = Filter hits received in RX FIFO Buffer
1110 = Filter hits received in RX Buffer 14
•
•
•
0001 = Filter hits received in RX Buffer 1
0000 = Filter hits received in RX Buffer 0

bit 11-8 **F14BP<3:0>:** RX Buffer mask for Filter 14 bits (same values as bits 15-12)

bit 7-4 **F13BP<3:0>:** RX Buffer mask for Filter 13 bits (same values as bits 15-12)

bit 3-0 **F12BP<3:0>:** RX Buffer mask for Filter 12 bits (same values as bits 15-12)

Register 21-14: CiRXFUL1: ECAN Receive Buffer Full Register 1

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXFUL15	RXFUL14	RXFUL13	RXFUL12	RXFUL11	RXFUL10	RXFUL9	RXFUL8
bit 15							bit 8

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXFUL7	RXFUL6	RXFUL5	RXFUL4	RXFUL3	RXFUL2	RXFUL1	RXFUL0
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0 **RXFUL<15:0>:** Receive Buffer n Full bits
1 = Buffer is full (set by module)
0 = Buffer is empty

Register 21-15: CiRXFUL2: ECAN Receive Buffer Full Register 2

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXFUL31	RXFUL30	RXFUL29	RXFUL28	RXFUL27	RXFUL26	RXFUL25	RXFUL24
bit 15							bit 8

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXFUL23	RXFUL22	RXFUL21	RXFUL20	RXFUL19	RXFUL18	RXFUL17	RXFUL16
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0　　**RXFUL<31:16>:** Receive Buffer n Full bits

　　　　　1 = Buffer is full (set by module)
　　　　　0 = Buffer is empty

Register 21-16: CiRXOVF1: ECAN Receive Buffer Overflow Register 1

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXOVF15	RXOVF14	RXOVF13	RXOVF12	RXOVF11	RXOVF10	RXOVF9	RXOVF8
bit 15							bit 8

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXOVF7	RXOVF6	RXOVF5	RXOVF4	RXOVF3	RXOVF2	RXOVF1	RXOVF0
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0　　**RXOVF<15:0>:** Receive Buffer n Overflow bits

　　　　　1 = Module attempted to write to a full buffer (set by module)
　　　　　0 = No overflow condition

Register 21-17: CiRXOVF2: ECAN Receive Buffer Overflow Register 2

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXOVF31	RXOVF30	RXOVF29	RXOVF28	RXOVF27	RXOVF26	RXOVF25	RXOVF24
bit 15							bit 8

R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0	R/C-0
RXOVF23	RXOVF22	RXOVF21	RXOVF20	RXOVF19	RXOVF18	RXOVF17	RXOVF16
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0　　**RXOVF<31:16>:** Receive Buffer n Overflow bits

　　　　　1 = Module attempted to write to a full buffer (set by module)
　　　　　0 = No overflow condition

Register 21-18: CiFCTRL: ECAN FIFO Control Register

R/W-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0	U-0
DMABS<2:0>			—	—	—	—	—

bit 15   bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FSA<4:0>				

bit 7   bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-13   **DMABS<2:0>:** DMA Buffer Size bits

                   111 = Reserved
                   110 = 32 buffers in DMA RAM
                   101 = 24 buffers in DMA RAM
                   100 = 16 buffers in DMA RAM
                   011 = 12 buffers in DMA RAM
                   010 = 8 buffers in DMA RAM
                   001 = 6 buffers in DMA RAM
                   000 = 4 buffers in DMA RAM

bit 12-5   **Unimplemented:** Read as '0'

bit 4-0   **FSA<4:0>:** FIFO Start Area bits

                   11111 = Read buffer RB31
                   11110 = Read buffer RB30
                   •
                   •
                   •
                   00010 = TX/RX buffer TRB2
                   00001 = TX/RX buffer TRB1
                   00000 = TX/RX buffer TRB0

Register 21-19:   CiFIFO: ECAN FIFO Status Register

U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
—	—	FBP<5:0>					
bit 15							bit 8

U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
—	—	FNRB<5:0>					
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-14   **Unimplemented:** Read as '0'

bit 13-8   **FBP<5:0>:** FIFO Buffer Pointer bits

011111 = RB31 buffer
011110 = RB30 buffer
•
•
•
000001 = TRB1 buffer
000000 = TRB0 buffer

bit 7-6   **Unimplemented:** Read as '0'

bit 5-0   **FNRB<5:0>:** FIFO Next Read Buffer Pointer bits

011111 = RB31 buffer
011110 = RB30 buffer
•
•
•
000001 = TRB1 buffer
000000 = TRB0 buffer

Register 21-20:   CiINTF: ECAN Interrupt Flag Register

U-0	U-0	R-0	R-0	R-0	R-0	R-0	R-0
—	—	TXBO	TXBP	RXBP	TXWAR	RXWAR	EWARN
bit 15							bit 8

R/C-0	R/C-0	R/C-0	U-0	R/C-0	R/C-0	R/C-0	R/C-0
IVRIF	WAKIF	ERRIF	—	FIFOIF	RBOVIF	RBIF	TBIF
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-14   **Unimplemented:** Read as '0'

bit 13   **TXBO:** Transmitter in Error State Bus Off bit
1 = Transmitter is in Bus Off state
0 = Transmitter is not in Bus Off state

bit 12   **TXBP:** Transmitter in Error State Bus Passive bit
1 = Transmitter is in Bus Passive state
0 = Transmitter is not in Bus Passive state

bit 11   **RXBP:** Receiver in Error State Bus Passive bit
1 = Receiver is in Bus Passive state
0 = Receiver is not in Bus Passive state

bit 10   **TXWAR:** Transmitter in Error State Warning bit
1 = Transmitter is in Error Warning state
0 = Transmitter is not in Error Warning state

bit 9   **RXWAR:** Receiver in Error State Warning bit
1 = Receiver is in Error Warning state
0 = Receiver is not in Error Warning state

bit 8   **EWARN:** Transmitter or Receiver in Error State Warning bit
1 = Transmitter or receiver is in Error Warning state
0 = Transmitter or receiver is not in Error Warning state

bit 7   **IVRIF:** Invalid Message Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 6   **WAKIF:** Bus Wake-up Activity Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 5   **ERRIF:** Error Interrupt Flag bit (multiple sources in CiINTF<13:8> register)
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

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

bit 3   **FIFOIF:** FIFO Almost Full Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 2   **RBOVIF:** RX Buffer Overflow Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 1   **RBIF:** RX Buffer Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 0   **TBIF:** TX Buffer Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

Register 21-21:   CiINTE: ECAN Interrupt Enable Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
IVRIE	WAKIE	ERRIE	—	FIFOIE	RBOVIE	RBIE	TBIE
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-8   **Unimplemented:** Read as '0'

bit 7   **IVRIE:** Invalid Message Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 6   **WAKIE:** Bus Wake-up Activity Interrupt Flag bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 5   **ERRIE:** Error Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

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

bit 3   **FIFOIE:** FIFO Almost Full Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 2   **RBOVIE:** RX Buffer Overflow Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 1   **RBIE:** RX Buffer Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

bit 0   **TBIE:** TX Buffer Interrupt Enable bit
1 = Interrupt request enabled
0 = Interrupt request not enabled

Register 21-22: CiVEC: ECAN Interrupt Code Register

U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
—	—	—			FILHIT<4:0>		
bit 15							bit 8

U-0	R-1	R-0	R-0	R-0	R-0	R-0	R-0
—				ICODE<6:0>			
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-13   **Unimplemented:** Read as '0'

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

`10000-11111` = Reserved
`01111` = Filter 15

•

•

•

`00001` = Filter 1
`00000` = Filter 0

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

bit 6-0   **ICODE<6:0>:** Interrupt Flag Code bits

`1000101-1111111` = Reserved
`1000100` = FIFO almost full interrupt
`1000011` = Receiver overflow interrupt
`1000010` = Wake-up interrupt
`1000001` = Error interrupt
`1000000` = No interrupt

•

•

•

`0100000-0111111` = Reserved
`0011111` = RB31 buffer interrupt
`0011110` = RB30 buffer interrupt

•

•

•

`0001001` = RB9 buffer interrupt
`0001000` = RB8 buffer interrupt
`0000111` = TRB7 buffer interrupt
`0000110` = TRB6 buffer interrupt
`0000101` = TRB5 buffer interrupt
`0000100` = TRB4 buffer interrupt
`0000011` = TRB3 buffer interrupt
`0000010` = TRB2 buffer interrupt
`0000001` = TRB1 buffer interrupt
`0000000` = TRB0 buffer interrupt

Register 21-23: CiCTRL1: ECAN Control Register 1

U-0	U-0	R/W-0	R/W-0	r-0	R/W-1	R/W-0	R/W-0
—	—	CSIDL	ABAT	—	REQOP<2:0>
bit 15							bit 8

R-1	R-0	R-0	U-0	R/W-0	U-0	U-0	R/W-0
\multicolumn{3}{l}{OPMODE<2:0>}	—	CANCAP	—	—	WIN
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	r = Bit is reserved
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 15-14	**Unimplemented:** Read as '0'
bit 13	**CSIDL:** Stop in Idle Mode bit
	1 = Discontinue module operation when device enters Idle mode
	0 = Continue module operation in Idle mode
bit 12	**ABAT:** Abort All Pending Transmissions bit
	1 = Signal all transmit buffers to abort transmission
	0 = Module will clear this bit when all transmissions are aborted
bit 11	**Reserved:** Do not use
bit 10-8	**REQOP<2:0>:** Request Operation Mode bits
	000 = Set Normal Operation mode
	001 = Set Disable mode
	010 = Set Loopback mode
	011 = Set Listen-Only mode
	100 = Set Configuration mode
	101 = Reserved
	110 = Reserved
	111 = Set Listen All Messages mode
bit 7-5	**OPMODE<2:0>:** Operation Mode bits
	000 = Module is in Normal Operation mode
	001 = Module is in Disable mode
	010 = Module is in Loopback mode
	011 = Module is in Listen-Only mode
	100 = Module is in Configuration mode
	101 = Reserved
	110 = Reserved
	111 = Module is in Listen All Messages mode
bit 4	**Unimplemented:** Read as '0'
bit 3	**CANCAP:** CAN Message Receive Timer Capture Event Enable bit
	1 = Enable input capture based on CAN message receive
	0 = Disable CAN capture
bit 2-1	**Unimplemented:** Read as '0'
bit 0	**WIN:** SFR Map Window Select bit
	1 = Use filter window
	0 = Use buffer window

Register 21-24: CiCTRL2: ECAN Control Register 2

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
—	—	—	DNCNT<4:0>				
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-5 **Unimplemented:** Read as '0'

bit 4-0 **DNCNT<4:0>:** DeviceNet™ Filter Bit Number bits

00000 = Do not compare data bytes
00001 = Compare bit 7 of data byte 0 with EID<17>
00010 = Compare bits <7:6> of data byte 0 with EID<17:16>
00011 = Compare bits <7:5> of data byte 0 with EID<17:15>
00100 = Compare bits <7:4> of data byte 0 with EID<17:14>
00101 = Compare bits <7:3> of data byte 0 with EID<17:13>
00110 = Compare bits <7:2> of data byte 0 with EID<17:12>
00111 = Compare bits <7:1> of data byte 0 with EID<17:11>
01000 = Compare bits <7:0> of data byte 0 with EID<17:10>
01001 = Compare bits <7:0> of data byte 0 and bit <7> of data byte 1 with EID<17:9>
01010 = Compare bits <7:0> of data byte 0 and bits <7:6> of data byte 1 with EID<17:8>
01011 = Compare bits <7:0> of data byte 0 and bits <7:5> of data byte 1 with EID<17:7>
01100 = Compare bits <7:0> of data byte 0 and bits <7:4> of data byte 1 with EID<17:6>
01101 = Compare bits <7:0> of data byte 0 and bits <7:3> of data byte 1 with EID<17:5>
01110 = Compare bits <7:0> of data byte 0 and bits <7:2> of data byte 1 with EID<17:4>
01111 = Compare bits <7:0> of data byte 0 and bits <7:1> of data byte 1 with EID<17:3>
10000 = Compare bits <7:0> of data byte 0 and bits <7:0> of data byte 1 with EID<17:2>
10001 = Compare bits <7:0> of byte 0 and bits <7:0> of byte 1 and bit <7> of byte 2 with EID<17:1>
10010 = Compare bits <7:0> of byte 0 and bits <7:0> of byte 1 and bits <7:6> of byte 2 with EID<17:0>
10011-11111 = Invalid selection

Register 21-25: CiTRmnCON: ECAN TX/RX Buffer m Control Register (m = 0,2,4,6; n = 1,3,5,7)

R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
TXENn	TXABTn	TXLARBn	TXERRn	TXREQn	RTRENn	TXnPRI<1:0>	
bit 15							bit 8

R/W-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
TXENm	TXABTm[1]	TXLARBm[1]	TXERRm[1]	TXREQm	RTRENm	TXmPRI<1:0>	
bit 7							bit 0

Legend:		C = Writable bit, but only '0' can be written to clear the bit	
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 15-8   Refer to definition for bits 7-0, Controls Buffer n

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

1 = Buffer TRBn is a transmit buffer
0 = Buffer TRBn is a receive buffer

bit 6   **TXABTm:** Message Aborted bit[1]

1 = Message was aborted
0 = Message completed transmission successfully

bit 5   **TXLARBm:** Message Lost Arbitration bit[1]

1 = Message lost arbitration while being sent
0 = Message did not lose arbitration while being sent

bit 4   **TXERRm:** Error Detected During Transmission bit[1]

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   **TXREQm:** Message Send Request bit

1 = Requests that a message be sent. The bit automatically clears when the message is successfully sent
0 = Clearing the bit to '0' while set requests a message abort

bit 2   **RTRENm:** Auto-Remote Transmit Enable bit

1 = When a remote transmit is received, TXREQ is set
0 = When a remote transmit is received, TXREQ is unaffected

bit 1-0   **TXmPRI<1:0>:** Message Transmission Priority bits

11 = Highest message priority
10 = High intermediate message priority
01 = Low intermediate message priority
00 = Lowest message priority

     Note 1: This bit is cleared when TXREQ is set.

Register 21-26: CiEC: ECAN Transmit/Receive Error Count Register

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
			TERRCNT<7:0>				
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
			RERRCNT<7:0>				
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-8   **TERRCNT<7:0>:** Transmit Error Count bits

bit 7-0   **RERRCNT<7:0>:** Receive Error Count bits

Register 21-27: CiRXD: ECAN Receive Data Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
			Received Data Word				
bit 15							bit 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
			Received Data Word				
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0   **DATA<15:0>:** Receive Data bits

Register 21-28: CiTXD: ECAN Transmit Data Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
			Transmit Data Word				
bit 15							bit 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
			Transmit Data Word				
bit 7							bit 0

Legend:	C = Writable bit, but only '0' can be written to clear the bit		
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 15-0   **DATA<15:0>:** Transmit Data bits

21.4   ECAN MESSAGE BUFFERS

ECAN message buffers are located in DMA RAM. They are not ECAN Special Function Registers. The user application must directly write into the DMA RAM area that is configured for ECAN message buffers. The location and size of the buffer area is defined by the user application.

This section provides information on how the message buffer words are organized for transmission and reception. (Refer to **21.2 "CAN Message Formats"** for message buffer layout details and **21.8 "DMA Controller Configuration"** for details on how to configure ECAN message buffers in DMA RAM.)

Buffer 21-1:   ECAN Message Buffer Word 0

U-x	U-x	U-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	SID10	SID9	SID8	SID7	SID6
bit 15							bit 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
SID5	SID4	SID3	SID2	SID1	SID0	SRR	IDE
bit 7							bit 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 15-13	**Unimplemented:** Read as '0'	
bit 12-2	**SID<10:0>:** Standard Identifier bits	
bit 1	**SRR:** Substitute Remote Request bit	
	When TXIDE = 0:	
	1 = Message will request remote transmission	
	0 = Normal message	
	When TXIDE = 1:	
	The SRR bit must be set to '1'	
bit 0	**IDE:** Extended Identifier bit	
	1 = Message will transmit extended identifier	
	0 = Message will transmit standard identifier	

Buffer 21-2:   ECAN Message Buffer Word 1

U-x	U-x	U-x	U-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	—	EID17	EID16	EID15	EID14
bit 15							bit 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
EID13	EID12	EID11	EID10	EID9	EID8	EID7	EID6
bit 7							bit 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 15-12	**Unimplemented:** Read as '0'
bit 11-0	**EID<17:6>:** Extended Identifier bits

Buffer 21-3: ECAN Message Buffer Word 2

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID5	EID4	EID3	EID2	EID1	EID0	RTR	RB1
bit 15							bit 8

U-x	U-x	U-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	RB0	DLC3	DLC2	DLC1	DLC0
bit 7							bit 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 15-10   **EID<5:0>:** Extended Identifier bits

bit 9   **RTR:** Remote Transmission Request bit

     When TXIDE = 1:
     1 = Message will request remote transmission
     0 = Normal message
     When TXIDE = 0:
     The RTR bit is ignored.

bit 8   **RB1:** Reserved Bit 1

     User must set this bit to '0' per CAN protocol.

bit 7-5   **Unimplemented:** Read as '0'

bit 4   **RB0:** Reserved Bit 0

     User must set this bit to '0' per CAN protocol.

bit 3-0   **DLC<3:0>:** Data Length Code bits

Buffer 21-4: ECAN Message Buffer Word 3

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
			Byte 1				
bit 15							bit 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
			Byte 0				
bit 7							bit 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 15-8   ECAN Message byte 1

bit 7-0   ECAN Message byte 0

Buffer 21-5:   **ECAN Message Buffer Word 4**

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
			Byte 3				
bit 15							bit 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
			Byte 2				
bit 7							bit 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 15-8   ECAN Message byte 3

bit 7-0   ECAN Message byte 2

Buffer 21-6:   **ECAN Message Buffer Word 5**

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
			Byte 5				
bit 15							bit 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
			Byte 4				
bit 7							bit 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 15-8   ECAN Message byte 5

bit 7-0   ECAN Message byte 4

Buffer 21-7:   **ECAN Message Buffer Word 6**

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
			Byte 7				
bit 15							bit 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
			Byte 6				
bit 7							bit 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 15-8   ECAN Message byte 7

bit 7-0   ECAN Message byte 6

Buffer 21-8:   ECAN Message Buffer Word 7

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	FILHIT4	FILHIT3	FILHIT2	FILHIT1	FILHIT0
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 7							bit 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 15-13   **Unimplemented:** Read as '0'

bit 12-8   **FILHIT<4:0>:** Filter Hit Code bits
Encodes number of filter that resulted in writing this buffer (only written by module for receive buffers, unused for transmit buffers).

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

21.5 ECAN OPERATING MODES

The ECAN module can operate in one of several modes selected by the user application. These modes include:

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

Operating modes are requested by the user application writing to the Request Operation mode (REQOP<2:0>) bits in ECAN Control Register 1 (CiCTRL1<10:8>). ECAN acknowledges entry into the requested mode by the Operation Mode (OPMODE<2:0>) bits (CiCTRL1<7:5>). Mode transition is performed in synchronization with the CAN network. That is, the ECAN controller waits until it detects a bus idle sequence (11 recessive bits) before it changes mode.

21.5.1 Configuration Mode

After hardware reset, the ECAN module is in the Configuration mode (OPMODE<2:0> = `100`). The error counters are cleared, and all registers contain the Reset values. In order to modify the ECAN bit time control registers (CiCFG1 and CiCFG2), the ECAN module must be in Configuration mode.

21.5.2 Normal Operation Mode

In Normal Operation mode, the ECAN module can transmit and receive CAN messages. Normal Operation mode is requested after initialization by programming the REQOP<2:0> bits (CiCTRL1<10:8>) to '000'. When OPMODE<2:0> = `000`, the module proceeds with normal operation.

21.5.3 Listen-Only Mode

Listen-Only mode is used mainly for bus monitoring without actually participating in the transmission process. The node in Listen-Only mode does not generate an acknowledge or error frames – one of the other nodes must do it. The Listen-Only mode can be used for detecting the baud rate on the CAN bus.

21.5.4 Listen All Messages Mode

Listen All Messages mode is used for system debugging. Basically, all messages are received, irrespective of their identifier, even when there is an error. If the Listen All Messages mode is activated, transmission and reception operate the same as Normal Operation mode except that if a message is received with an error, it is still transferred to message buffer.

21.5.5 Loopback Mode

Loopback mode is used for self test to allow the ECAN module to receive its own message. In this mode, the ECAN transmit path is connected to the receive path internally. A "dummy" Acknowledge is provided, thereby eliminating the need for another node to provide the Acknowledge bit.

21.5.6 Disable Mode

Disable mode is used to ensure a safe shutdown before putting the device in Sleep or Idle mode. That is, the ECAN controller waits until it detects a bus idle sequence (11 recessive bits) before it changes mode. When the module is in Disable mode, it stops its own clocks, having no effect on the CPU or other modules. The module wakes up when bus activity occurs or when the CPU sets OPMODE<2:0> to '000'.

The CiTX pin stays in the recessive state while the module is in Disable mode.

21.6 TRANSMITTING ECAN MESSAGES

A node originating a message is a transmitter of that message. The node remains a transmitter until the bus becomes idle or the unit loses arbitration. Figure 21-8 illustrates a typical ECAN transmission process.

Message buffers 0-7 (located in DMA RAM) are configured to transmit or receive CAN messages using the TX/RX Buffer Selection (TXENn) bit in the corresponding ECAN TX/RX Buffer m Control Register (CiTRmnCON<7>). If the TXENn bit is set, the message buffer is configured for transmission. See **21.2 "CAN Message Formats"** for the layout of standard and extended frames in the message buffer and the states of IDE, SRR, RTR, RB0 and RB1 bits for Standard Data, Extended Data, Standard Remote or Extended remote frames as per the CAN protocol.

21.6.1 Message Transmission Flow

To transmit a message over the CAN bus, the user application must perform these tasks:

- Configure a message buffer for transmission and assign a priority to the buffer
- Write the CAN message in the message buffer located in DMA RAM
- Set the transmit request bit for the buffer to initiate message transmission

Message transmission is initiated by setting the Message Send Request (TXREQm) bit in ECAN Transmit/Receive Control Register (CiTRmnCON<3>). The TXREQm bit is cleared automatically after the message is transmitted. Before the SOF is sent, all the buffers ready for transmission are examined to determine which buffer has the highest priority. The transmit buffer with the highest priority is sent first.

Each of the transmit message buffers can be assigned to any of the four user-application defined priority levels using CiTRmnCON<TXnPRI> bits.

TXnPRI<1:0> Message Transmission Priority selections are:

- `11` = The transmit message has the highest priority
- `10` = The transmit message has intermediate high priority
- `01` = The transmit message has intermediate low priority
- `00` = The transmit message has the lowest priority

There is a natural order of priority for message buffers that are assigned to the same user-application defined priority level. Message buffer 7 has the highest natural order of priority. User-application defined priority levels override the natural order of priority.

Figure 21-8: ECAN™ Transmission
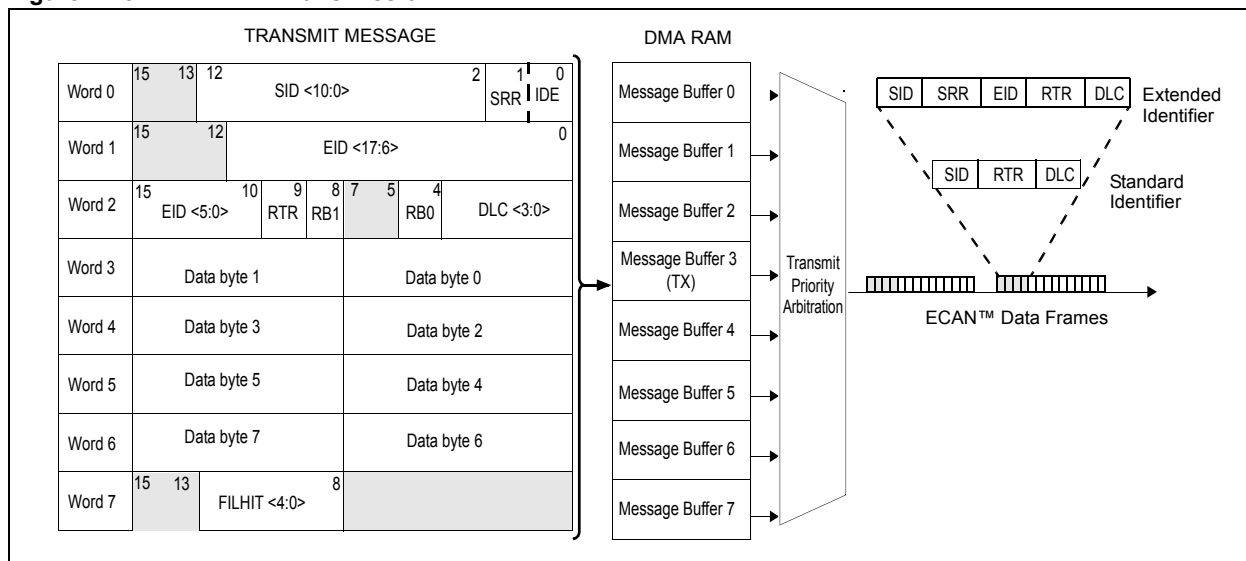
Example 21-1 illustrates code to transmit a standard frame using message buffer 0.

Example 21-1:   Code Example for Standard Data Frame Transmission

```
/* Assign 32x8word Message Buffers for ECAN1 in DMA RAM */

unsigned int ecan1MsgBuf[32][8] __attribute__ (space(dma));
DMA1STA = __builtin_dmaoffset(ecan1MsgBuf);

/* Configure Message Buffer 0 for Transmission and assign priority*/

C1TR01CONbits.TXEN0 = 0x1;
C1TR01CONbits.TX0PRI = 0x3;

/* WRITE TO MESSAGE BUFFER 0 */
/* CiTRBnSID = 0bxxx1 0010 0011 1100
IDE = 0b0
SRR = 0b0
SID<10:0>= 0b100 1000 1111 */

ecan1MsgBuf[0][0] = 0x123C;
/* CiTRBnEID = 0bxxxx 0000 0000 0000
EID<17:6> = 0b0000 0000 0000 */

ecan1MsgBuf[0][1] = 0x0000;

/* CiTRBnDLC = 0b0000 0000 xxx0 1111
EID<17:6> = 0b000000
RTR = 0b0
RB1 = 0b0
RB0 = 0b0
DLC = 0b1000 */

ecan1MsgBuf[0][2] = 0x8;

/* WRITE MESSAGE DATA BYTES */

ecan1MsgBuf[0][3] = 0xabcd;
ecan1MsgBuf[0][4] = 0xabcd;
ecan1MsgBuf[0][5] = 0xabcd;
ecan1MsgBuf[0][6] = 0xabcd;

/* REQUEST MESSAGE BUFFER 0 TRANSMISSION */

C1TR01CONbits.TXREQ0 = 0x1;
```

Example 21-2 illustrates code to transmit an extended frame using message buffer 2.

Example 21-2:   Code Example for Extended Data Frame Transmission

```
/* Assign 32x8word Message Buffers for ECAN1 in DMA RAM */
unsigned int ecan1MsgBuf[32][8] __attribute__(space(dma));
DMA1STA = __builtin_dmaoffset(ecan1MsgBuf);

/* Configure Message Buffer 2 for Transmission and assign priority*/

C1TR23CONbits.TXEN2 = 0x1;
C1TR23CONbits.TX2PRI = 0x2;

/* WRITE TO MESSAGE BUFFER 2 */
/* CiTRBnSID = 0bxxx1 0010 0011 1101
IDE = 0b0
SRR = 0b0
SID<10:0> : 0b100 1000 1111 */

ecan1MsgBuf[2][0] = 0x123D;

/* CiTRBnEID = 0bxxxx 1111 0000 0000
EID<17:6> = 0b1111 0000 0000 */

ecan1MsgBuf[2][1] = 0x0F00;

/* CiTRBnDLC = 0b0000 1100 xxx0 1111
EID<17:6> = 0b000011
RTR = 0b0
RB1 = 0b0
RB0 = 0b0
DLC = 0b1000 */

ecan1MsgBuf[2][2] = 0x0C08;

/* WRITE MESSAGE DATA BYTES */

ecan1MsgBuf[2][3] = 0xabcd;
ecan1MsgBuf[2][4] = 0xabcd;
ecan1MsgBuf[2][5] = 0xabcd;
ecan1MsgBuf[2][6] = 0xabcd;

/* REQUEST MESSAGE BUFFER 2 TRANSMISSION */

C1TR23CONbits.TXREQ2 = 0x1;
```

21.6.2   Aborting a Transmit Message

Setting the Abort All Pending Transmissions (ABAT) bit in ECAN Control Register 1 (CiCTRL1<12>) requests an abort of all pending messages. To abort a specific message, the Message Send Request (TXREQm) bit (CiTRmnCON<3>) associated with that message buffer must be cleared. In either case, the message is only aborted if the ECAN module has not started transmitting the message on the bus.

21.6.3 Remote Transmit Request/Response

21.6.3.1 REMOTE TRANSMIT REQUEST

A node expecting to receive a data frame with a specific identifier value can initiate the transmission of the respective data by another node by sending the remote frame. The remote frame can be either in a Standard or Extended format.

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)

To transmit a remote frame, the user application must perform these tasks:

• Configure the message buffer for transmission and assign a priority to the buffer
• Write the remote frame in the appropriate message buffer. The transmitted identifier must be identical to the identifier of the data frame to be received
• Set the transmit request bit for the buffer to initiate transmission of the remote frame
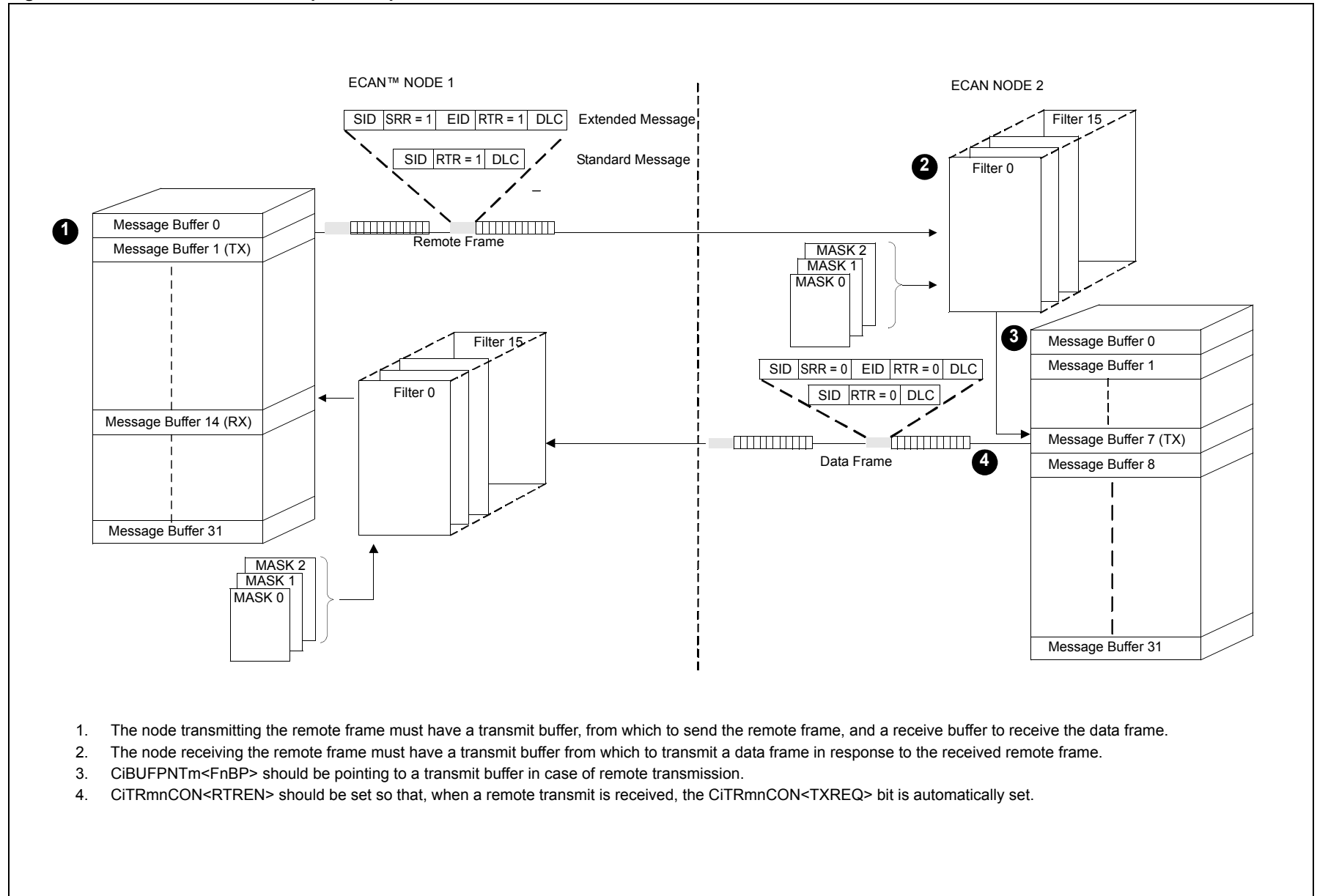
21.6.3.2 REMOTE TRANSMIT RESPONSE

The node that is acting as the source to respond to the remote frame request needs to configure an acceptance filter to match the identifier of the Remote Request Frame. Message buffers 0-7 can respond to remote requests; therefore, the Acceptance Filter Buffer Pointer (FnBP) should point to one of the 8 message buffers. The TX/RX Buffer Selection (TXENn) and Auto-Remote Transmit Enable (RTRENn) bits in the ECAN Transmit/Receive Control Register (CiTRmnCON) must be set to respond to the Remote Request Frame.

This is the only case where the Acceptance Filter Buffer Pointer (FnBP) points to a message buffer that is configured for transmission (TXENn = 1).

Figure 21-9 illustrates the remote frame handling process:

1. ECAN Node 1 sends a Remote Transmit Request (using message buffer 1).
2. ECAN Node 2 receives the request and responds by sending the data frame (using message buffer 7).
3. The data frame is received by ECAN Node 1.
4. The data frame is stored in message buffer 14 of ECAN Node 1.

Figure 21-9:   Remote Transmit Request/Response

1. The node transmitting the remote frame must have a transmit buffer, from which to send the remote frame, and a receive buffer to receive the data frame.
2. The node receiving the remote frame must have a transmit buffer from which to transmit a data frame in response to the received remote frame.
3. CiBUFPNTm<FnBP> should be pointing to a transmit buffer in case of remote transmission.
4. CiTRmnCON<RTREN> should be set so that, when a remote transmit is received, the CiTRmnCON<TXREQ> bit is automatically set.

Example 21-3 illustrates the code required to transmit an extended remote frame using message buffer 2.

Example 21-3:   Code Example for Transmitting Extended Remote Frame

```
/* Assign 32x8word Message Buffers for ECAN1 in DMA RAM */

unsigned int ecan1MsgBuf[32][8] __attribute__(space(dma));
DMA1STA = __builtin_dmaoffset(ecan1MsgBuf);

/* Configure Message Buffer 0 for Transmission and assign priority*/

C1TR23CONbits.TXEN0 = 0x1;
C1TR23CONbits.TX0PRI = 0x2;

/* WRITE TO MESSAGE BUFFER 0 */
/* CiTRBnSID = 0bxxx1 0010 0011 1111
IDE = 0b1
SRR = 0b1
SID<10:0> : 0b100 1000 1111 */

ecan1MsgBuf[2][0] = 0x123F;

/* CiTRBnEID = 0bxxxx 1111 0000 0000
EID<17:6> = 0b1111 0000 0000 */

ecan1MsgBuf[2][1] = 0x0F00;

/* CiTRBnDLC = 0b0000 1110 xxx0 1111
EID<17:6> = 0b000011
RTR = 0b1
RB1 = 0b0
RB0 = 0b0
DLC = 0b0 */

ecan1MsgBuf[2][2] = 0x0E00;

/* THERE ARE NO DATA BYTES FOR A REMOTE MESSAGE */
/* REQUEST MESSAGE BUFFER 0 TRANSMISSION */

C1TR23CONbits.TXREQ2 = 0x1;
```

21.7 RECEIVING ECAN MESSAGES

The ECAN module can receive both standard and extended frames on the CAN bus node. Moreover, it has the additional capability of automatically transferring received messages to user-defined buffers in DMA RAM, thereby eliminating the need for the user application to explicitly copy messages from hardware registers to user-defined buffers. The storage format of each message within the DMA buffer is identical to that of transmit buffers, with each message (including the associated status register) occupying 8 words in DMA RAM.

The two main stages that constitute the ECAN reception process are described in the following section. Figure 21-10 and Figure 21-13 are examples of a simplified reception process.

21.7.1 Message Reception and Acceptance Filtering

As shown in Figure 21-10, every incoming message on the bus is received into a Message Assembly Buffer, and its identifier field is compared with a set of 16 user-defined acceptance filters. Each received standard data frame contains an 11-bit Standard Identifier (SID), and each extended data frame contains an 11-bit SID, and an 18-bit Extended Identifier (EID). If all bits in the incoming identifier completely match the corresponding bits in any of the acceptance filters, the ECAN module generates a DMA transfer request to the DMA Controller so that the message can be received into the appropriate buffer in DMA RAM.

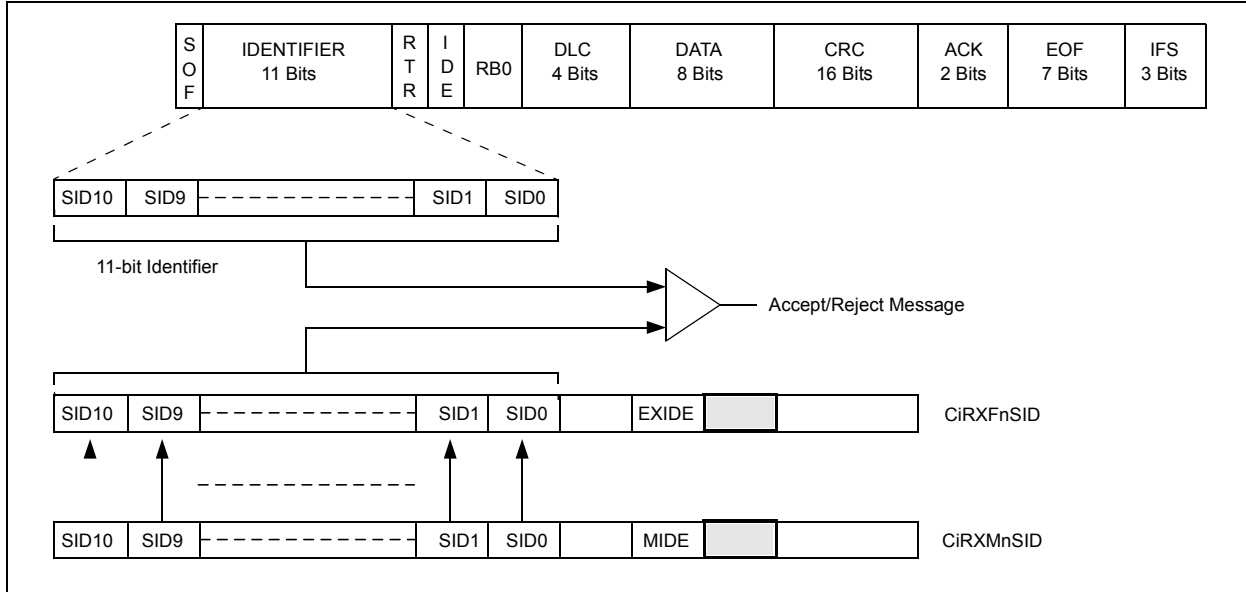**Figure 21-10: Message Reception and Acceptance Filtering**
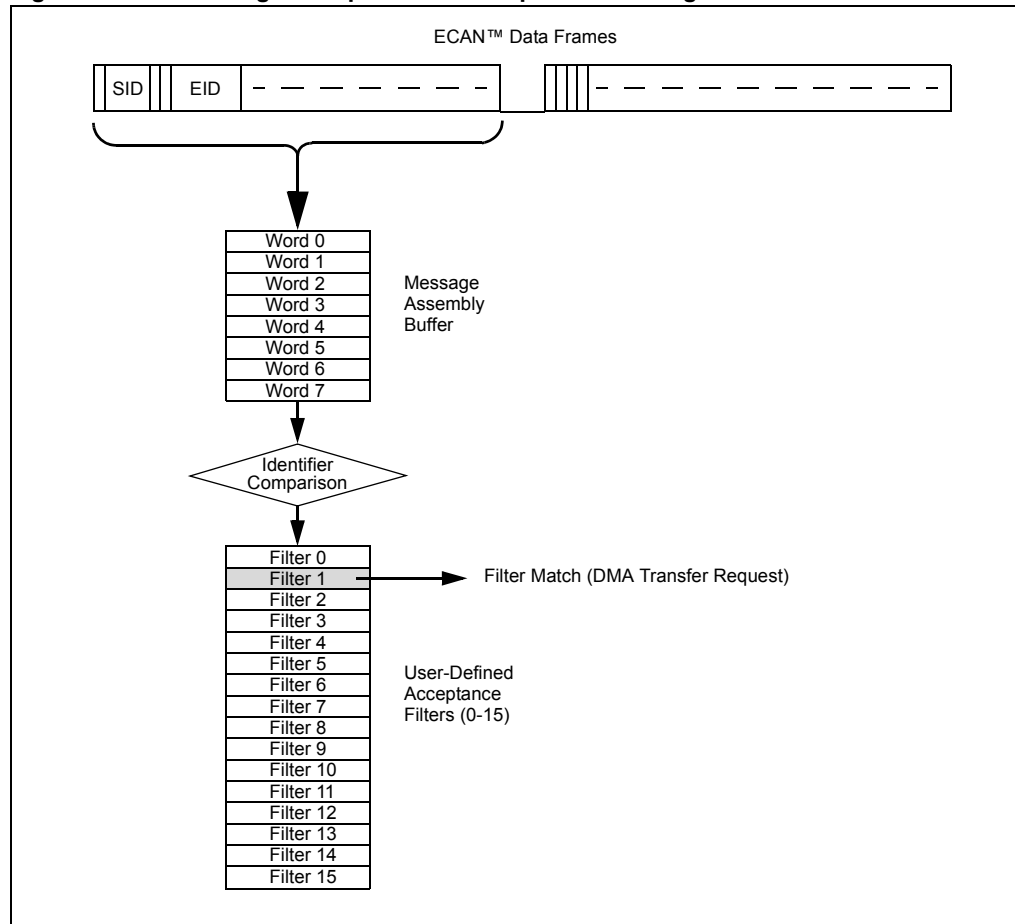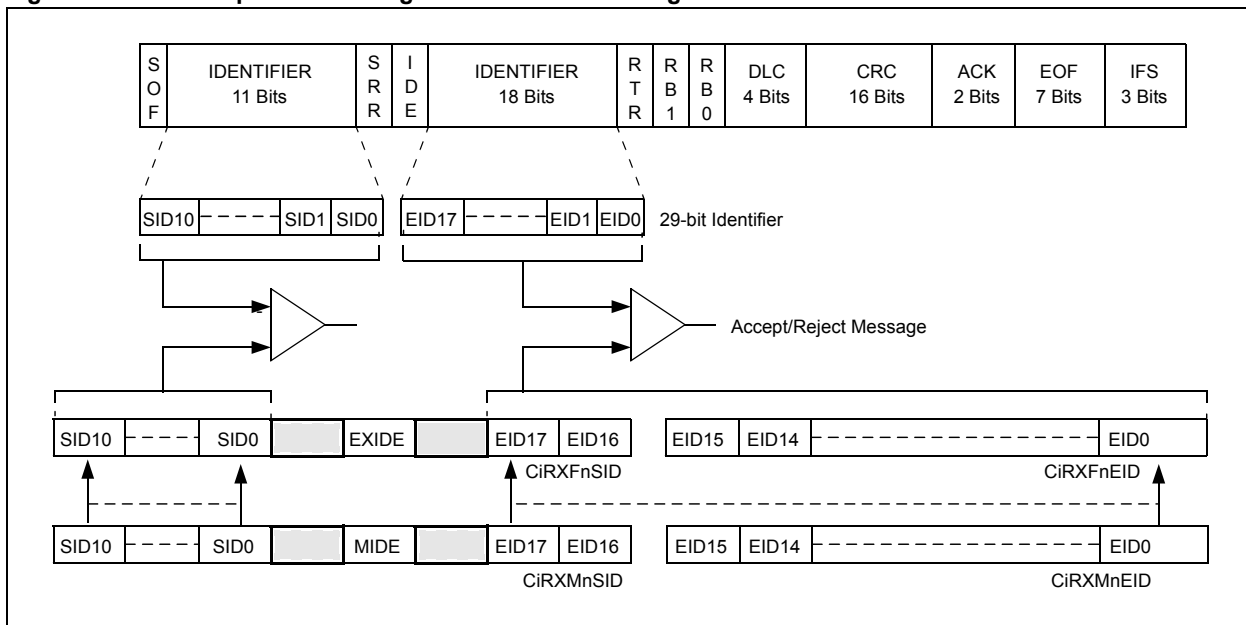
21.7.1.1 ACCEPTANCE FILTERS

Figure 21-11 shows the incoming message identifier in comparison with filter/mask bits for standard frames. Figure 21-12 shows the incoming message identifier in comparison with filter/mask bits for extended frames.

Figure 21-11: Acceptance Filtering for a Standard Message

Figure 21-12: Acceptance Filtering for a Extended Message

Acceptance filters 0-15 can be individually enabled or disabled using the Enable Filter (FLTENn) bits in the ECAN Acceptance Filter Enable (CiFEN1<15:0>) register. The value of 'n' signifies the register bit and corresponds to the index of the acceptance filter.

Acceptance filters 0-15 specify identifiers that must be contained in an incoming message for its contents to be passed to a receive buffer. Each of these filters consists of two registers – one for Standard Identifiers and the other for Extended Identifiers. These registers are identified as:

- **CiRXFnSID:** ECAN Acceptance Filter n Standard Identifier register (where n = 0-15)
- **CiRXFnEID:** ECAN Acceptance Filter n Extended Identifier register (where n = 0-15)

21.7.1.2   ACCEPTANCE FILTER MASKS

As shown in Figure 21-11 and Figure 21-12, an acceptance filter mask determines which bits in the incoming message identifiers are examined with the acceptance filters.

Acceptance filters optionally select one of the acceptance filter masks using the Mask Source Select (FnMSK<1:0>) mask select bits in the CiFMSKSEL1 and CiFMSKSEL2 registers:

- **CiFMSKSEL1<FnMSK>:** Mask Selection for Filters 0-7
- **CiFMSKSEL2<FnMSK>:** Mask Selection for Filters 8-15

The selection values for the FnMSK<1:0> bit are shown in Table 21-1.

Table 21-1:   FnMSK<1:0> Selections and Values

Value	Selection
00	Select acceptance filter mask 0
01	Select acceptance filter mask 1
10	Select acceptance filter mask 2
11	Reserved

Table 21-2 is a truth table that indicates how each bit in the identifier is compared to the masks and filters to determine if the message should be accepted or rejected. The mask bit essentially determines which bits to apply the filter to. If any mask bit is set to zero, that bit is automatically accepted, regardless of the filter bit.

Table 21-2:   Acceptance Filter/Mask Truth Table

Mask (SIDn/EIDn)	Filter (SIDn/EIDn)	Message (SIDn/EIDn)	Accept or Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

21.7.1.3   MESSAGE TYPE SELECTION

The Extended Identifier Enable (EXIDE) bit in the ECAN Acceptance Filter n Standard Identifier (CiRXFnSID<3>) register enables reception of either Standard Identifier or Extended Identifier messages. The Identifier Receive Mode (MIDE) bit in the ECAN Acceptance Filter Mask n Standard Identifier (CiRXMnSID<3>) register enables the CiRXFnSID<EXIDE> bit. If the CiRXMnSID<MIDE> bit is set, only the type of message selected by the CiRXFnSID<EXIDE> bit is accepted. If CiRXMnSID<MIDE> is clear, the CiRXFnSID<EXIDE> bit is ignored and all messages that match the filter are accepted.

Table 21-3 shows the bit settings and resulting selection.

Table 21-3:   EXIDE and MIDE Selections

EXIDE	MIDE	Selection
0	1	Acceptance filter to check for standard identifier.
1	1	Acceptance filter to check for extended identifier.
x	0	Acceptance filter to check for standard/extended identifier.

21.7.1.4   ACCEPTANCE FILTER CONFIGURATION

Example 21-4 illustrates the code to configure acceptance filter 0 to receive Standard Identifier messages using the acceptance filter mask register to mask SID<2:0> bits.

Example 21-4:   Code Example for Filtering Standard Data Frame

```
/* Enable Window to Access Acceptance Filter Registers */
C1CTRL1bits.WIN=0x1;

/* Select Acceptance Filter Mask 0 for Acceptance Filter 0 */
C1FMSKSEL1bits.F0MSK=0x0;

/* Configure Acceptance Filter Mask 0 register to mask SID<2:0>
Mask Bits (11-bits) : 0b000 0000 0111 */
C1RXM0SIDbits.SID = 0x0007;

/* Configure Acceptance Filter 0 to match standard identifier
Filter Bits (11-bits): 0b011 1010 xxx */
C1RXF0SIDbits.SID = 0x01D0;

/* Acceptance Filter 0 to check for Standard Identifier */
C1RXM0SIDbits.MIDE = 0x1;
C1RXF0SIDbits.EXIDE= 0x0;

/* Acceptance Filter 0 to use Message Buffer 10 to store message */
C1BUFPNT1bits.F0BP = 0xA;

/* Filter 0 enabled for Identifier match with incoming message */
C1FEN1bits.FLTEN0=0x1;

/* Clear Window Bit to Access ECAN Control Registers */
C1CTRL1bits.WIN=0x0;
```

Example 21-5 illustrates the code to configure acceptance filter 2 to receive Extended Identifier messages using the acceptance filter mask register to mask EID<5:0> bits.

Example 21-5:   Code Example for Filtering Extended Data Frame

```
/* Select Acceptance Filter Mask 1 for Acceptance Filter 2 */

C1FMSKSEL1bits.F2MSK=0x1;

/* Configure Acceptance Filter Mask 1 register to mask EID<5:0>
Mask Bits (29-bits) : 0b0 0000 0000 0000 0000 0000 0011 1111
SID<10:0> : 0b00000000000 ..SID<10:0> or EID<28:18>
EID<17:16> : 0b00 ..EID<17:16>
EID<15:0> : 0b0000000000111111 ..EID<15:0> */

C1RXM1SIDbits.SID = 0x0;
C1RXM1SIDbits.EID = 0x0;
C1RXM1EIDbits.EID = 0x3F;

/* Configure Acceptance Filter 2 to match extended identifier
Filter Bits (29-bits) : 0b1 1110 0000 0011 1111 0101 10xx xxxx
SID<10:0> : 0b11110000000 ..SID<10:0> or EID<28:18>
EID<17:16> : 0b11 ..EID<17:16>
EID<15:0> : 0b1111010110xxxxxx ..EID<15:0> */

C1RXF2SIDbits.SID = 0x780;
C1RXF2SIDbits.EID = 0x3;
C1RXM2EIDbits.EID = 0xF680;

/* Acceptance Filter 2 to check for Extended Identifier */

C1RXM1SIDbits.MIDE = 0x1;
C1RXF2SIDbits.EXIDE= 0x1;

/* Acceptance Filter 2 to use Message Buffer 5 to store message */

C1BUFPNT1bits.F2BP = 0x6;

/* Enable Acceptance Filter 2 */

C1FEN1bits.FLTEN1=0x1;
```

21.7.2 Buffer Selection and DMA Transfer

As shown in Figure 21-13, if a filter match occurs, a DMA transfer request is generated by the ECAN module to the DMA Controller to automatically copy the received message into the appropriate message buffer in a user-defined DMA RAM area. The ECAN module supports up to 32 message buffers. The user application can use DMA Buffer Size (DMABS<2:0>) bits in the ECAN FIFO Control (CiFCTRL<15:13>) register to select either 4, 6, 8, 12, 16, 24 or 32 message buffers. The selection of the receive buffer index (and therefore the DMA RAM addresses in which a message is written by the DMA Controller), is dependent on which filter matched the incoming identifier, and is user-configurable. The DMA Controller moves the data into the appropriate addresses in the DMA RAM area and generates a DMA interrupt after the user-specified number of words are transferred. Please refer to **21.8 "DMA Controller Configuration"** for more details on DMA channel configuration for ECAN data transfers.

Figure 21-13: Buffer Selection and DMA Transfer
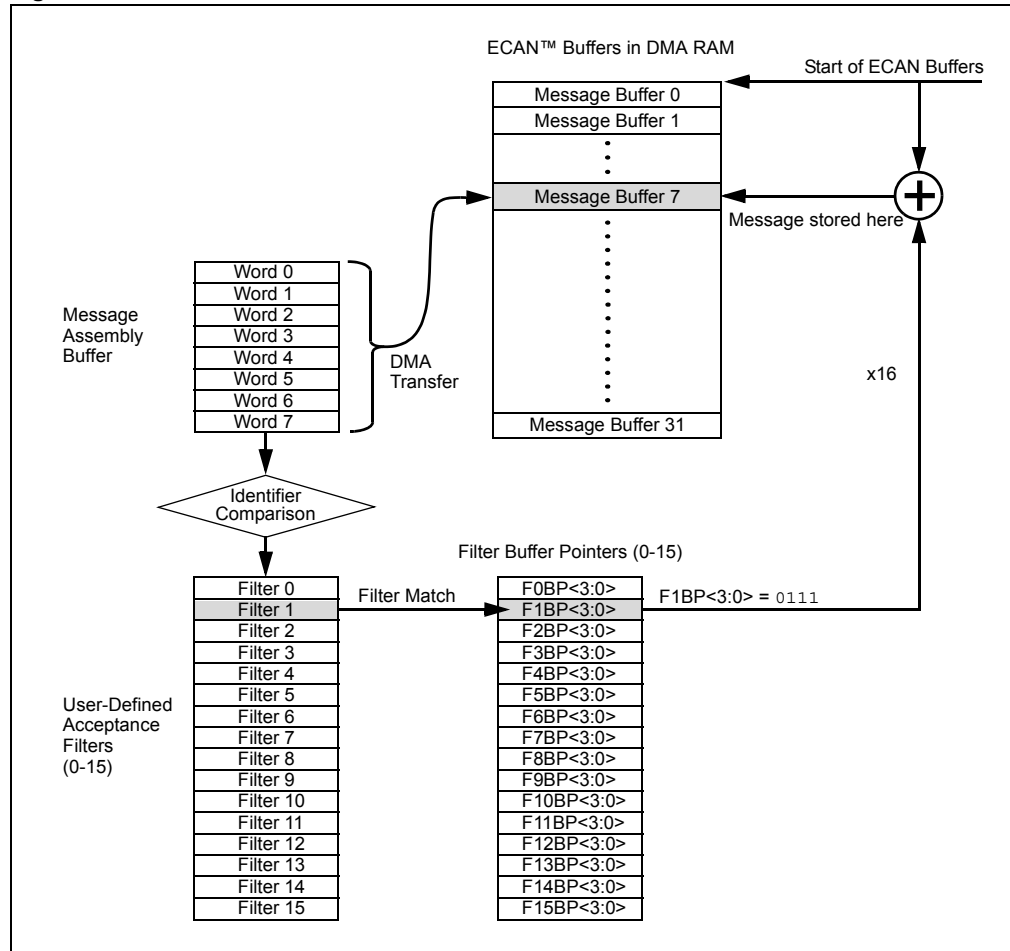
21.7.2.1   BUFFER SELECTION

There are four Acceptance Filter Buffer Pointer registers that select which message buffer the received message is stored into for acceptance filters 0 to 15.

- **CiBUFPNT1<FnBP>:** Buffer pointer for acceptance filters 0-3
- **CiBUFPNT2<FnBP>:** Buffer pointer for acceptance filters 4-7
- **CiBUFPNT3<FnBP>:** Buffer pointer for acceptance filters 8-11
- **CiBUFPNT4<FnBP>:** Buffer pointer for acceptance filters 12-15

When the incoming message identifier is matched by one of the acceptance filters, the internal logic looks up the buffer pointer (FnBP<3:0>) and uses that as an address for the corresponding message buffer. The address is provided to the DMA channel by the peripheral. Hence the DMA channel must be configured in Peripheral Indirect mode.

The values for FnBP<3:0> are interpreted as follows:

0000   Message is received in message buffer 0

0001   Message is received in message buffer 1

•

•

•

1110   Message is received in message buffer 14

1111   Message is received in Receive FIFO Buffer

> **Note:** Multi-message buffering can be implemented by a user application by configuring multiple acceptance filters with the same value. In this case, a received message may match multiple filters, and the ECAN module will assign the message to the lowest-numbered matching filter pointing to an empty buffer.

21.7.2.2   RECEIVING MESSAGES INTO MESSAGE BUFFERS 0-7

Message buffers 0-7 can be configured to transmit or receive CAN messages using the TX/RX Buffer Selection (TXENm) bit in the ECAN TX/RX Buffer m Control Register (CiTRmnCON<7>). The Acceptance Filter Buffer Pointer (FnBP) selects one of the message buffers to store the received message, provided it is configured as a receive buffer (TXENm = 0).

If a message buffer is set up as a transmitter with the CiTRmnCON<RTRENn> bit set, and an acceptance filter pointing to that message buffer detects a message, the message buffer will handle the RTR rather than store the message. This is the only case where the Acceptance Filter Buffer Pointer (FnBP) points to a message buffer that is configured for transmission (TXENn = 1).

> **Note:** The user application should not set the Transmit Request (TXREQ) bit in the CiTRmnCON register when the buffer is configured for receive operation (TXEN = 0). This could result in unpredictable module behavior.

21.7.2.3   RECEIVING MESSAGES INTO MESSAGE BUFFERS 8-14

Message buffers 8-14 are receive buffers. The Acceptance Filter Buffer Pointer (FnBP) determines which message buffer to use.

21.7.2.4   RECEIVING MESSAGES INTO MESSAGE BUFFERS 15-31

Message buffers 15-31 are receive buffers and are only usable as FIFO buffers because the Acceptance Filter Buffer Pointer (FnBP<3:0>) bits can only directly address 16 entities. When FnBP<3:0> = 1111, the results of a hit on that filter will write to the next available buffer location within the FIFO.

21.7.2.5 RECEIVE BUFFER STATUS BITS

The receive buffers contain two status bits per message buffer, the Message Buffer Full Flag (RXFULn) and the Message Buffer Overflow Flag (RXOVFn). These status bits are grouped into registers for buffer full status and buffer overflow status.

- **CiRXFUL1<RXFULn>:** Receive buffer full status for message buffers 0-15
- **CiRXFUL2<RXFULn>:** Receive buffer full status for message buffers 16-31
- **CiRXOVF1<RXOVFn>:** Receive buffer overflow status for message buffers 0-15
- **CiRXOVF2<RXOVFn>:** Receive buffer overflow status for message buffers 16-31

When a received message is stored into a message buffer, the respective Buffer Full Flag (RXFULn) is set in the Receive Buffer Full register, and a received buffer interrupt (CiINTF<RBIF>) is generated. If an incoming message caused a filter match, and the message buffer assigned to the matching filter is full (i.e., the RXFULn bit associated with that buffer is already '1'), the corresponding RXOVFn bit (where 'n' is the number of the message buffer associated with that buffer) is set and a received buffer overflow interrupt is generated (CiINTF<RBOVIF>). The message is lost.

> **Note:** If multiple filters match the identifier of the incoming message, and all the message buffers assigned to all the matching filters are full, the RXOVFn bit corresponding to the lowest numbered matching filter is set.

21.7.3 FIFO Buffer Operation

The ECAN module supports up to 32 message buffers. The user application can employ the DMA Buffer Size (DMABS<2:0>) bits in the ECAN FIFO Control (CiFCTRL) register to specify 4, 6, 8, 12, 16, 24 or 32 message buffers. The FIFO Start Area (FSA<4:0>) bits (CiFCTRL<4:0>) are used to specify the start of the FIFO within the buffer area. The end of FIFO is based on the number of message buffers defined in the DMABS<2:0> bits.

The user application should not allocate a FIFO area that contains transmit buffers. Should this condition occur, the module will attempt to point to the transmit buffer, but when a message is received for that buffer, an overflow condition causes the message contents to be lost.

Figure 21-14 shows that one of the message acceptance filters is set to store a received message in FIFO (FnBP = 1111). The start of FIFO is set to message buffer 5 (CiFCTRL<FSA> = 00101) and the end of FIFO is set to message buffer 11 (CiFCTRL<DMABS> = 011) by allocating 12 message buffers.

Figure 21-14: Receiving Messages in FIFO

1. The Acceptance Filter Buffer Pointer 'FnBP' should be '111' to Store Received Message in FIFO.
2. The starting address of FIFO is specified by CiFCTRL<FSA>. In the example above, FSA <4:0> = '00101'.
3. End address of FIFO is given by CiFCTRL<DMABS>. In the example above, DMABS <2:0> = '011'.
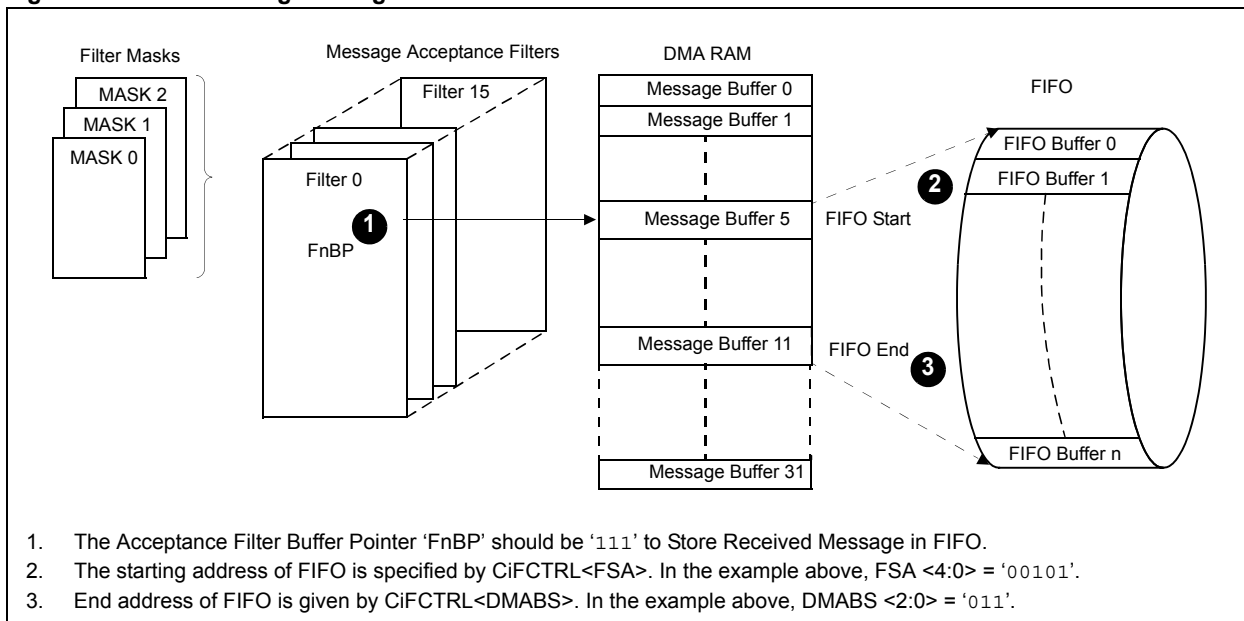
21.7.3.1 RECEIVING MESSAGES INTO FIFO AREA

The acceptance filter stores the received message in the FIFO area when FnBP<3:0> = 1111. It uses a simple buffer pointer, beginning with the start of the FIFO as defined above, and incrementing sequentially through the set of buffers within the FIFO area. When the end of the buffer is reached, the counter wraps and points to the start of FIFO area.

The write pointer value is accessible and (only) readable by the application software in the CiFIFO<FBP> bits. When the message is stored in the buffer, the RXFUL bit associated with the buffer is set, and the FIFO buffer counter increments.

If the FBP<5:0> value points to a buffer, and the RXFUL bit associated with that buffer is already '1' at the time of the filter hit and before writing of the message contents, the RXOVL bit associated with that buffer is set and the message is lost. After the message is lost, the FBP<5:0> value increments normally.

If the FBP<5:0> value points to a transmit/receive buffer that is selected as a transmit buffer at the time of the filter hit and before writing of the message contents, the RXOVL bit associated with that buffer is set and the message is lost. After the message is lost, the FBP<5:0> value increments normally.

The application software unloads the FIFO by reading the contents of a buffer. Once the buffer location is read, the application software clears the RXFUL bit corresponding to that buffer. When an RXFUL bit is cleared, the number of that corresponding buffer, plus one, is written to the CiFIFO <FNRB> bits by the module. Application software can (only) read this value, left shift it by 4 bits and use it as an address offset for the next buffer to be read. The application software should read the buffers sequentially.

The module generates an interrupt condition if the FIFO is about to fill. This condition is computed mathematically, as shown in Equation 21-1.

Equation 21-1: FIFO Interrupt Calculation

$$FNRB - FBP = 1$$
$$\text{or}$$
$$(FNRB = START) \ AND \ (FBP = END)$$

The interrupt is generated as the RXFUL bit is set for the buffer just written and after the FBP bit is updated. The computation uses the updated FBP value.

21.7.4 FIFO Example

Figure 21-15 illustrates seven examples of FIFO operation. The cases illustrated assume that Start of FIFO is set to message buffer 5 (CiFCTRL<FSA> = 101) and End of FIFO is set to message buffer 11 (CiFCTRL<DMABS> = 011).

- **Case 1** is the initialized case of the FIFO before any messages are received. The FIFO Buffer Pointer points to message buffer 5 (FRB = 5), and the FIFO Next Read Buffer Pointer points to message buffer 6 (FNRB = 5).

- **Case 2** shows the FIFO after one message is received and transferred to message buffer 5. The FIFO Buffer Pointer is incremented (FBP = 6), and the RXFUL status bit for message buffer 5 is set (RXFUL = 1).

- **Case 3** shows the FIFO after the sixth received message. The FIFO Buffer Pointer points to the last location in the FIFO area (FBP = 5 + 6 = 11), and the FIFO Next Read Pointer points to start of FIFO (FNRB = 5). In this case, FIFO is almost full and generates a FIFO interrupt.

- **Case 4** shows the FIFO after the application software reads the first received message. When the application software clears the RXFUL status bit for message buffer 5, the module writes the FIFO Next Read Buffer Pointer with MB5 plus 1 (FNRB = 5 + 1 = 6).

- **Case 5** shows the FIFO after the seventh message is received and written to message buffer 11. The RXFUL status bit for message buffer 11 is set (RXFUL = 1). Instead of incrementing, the FIFO Buffer Pointer is reloaded with the FIFO Start address (FBP = FSA = 5). Note that FBP is now mathematically one less than FNRB, which is the condition that generates the FIFO interrupt at the time the RXFUL status bit is set for message buffer 11.

- **Case 6** shows the FIFO after the eighth message is received and written to message buffer 5. Now the FIFO is full. There is no interrupt signaled for this condition.
- **Case 7** shows the FIFO after the ninth received message. Now the FIFO has overflowed. The module sets the RXOVL bit for the buffer intended for writing. The message is lost. The module generates a receive overflow interrupt.

Figure 21-15: Example of FIFO Operation

Legend: Shaded message buffers indicate the presence of a received message ready to be read by the user application.
MB n represents message buffers 5-11.
Write signifies that a message is stored in the FIFO message buffer and the RXFUL flag associated with that buffer is set.
Read signifies that the application software unloads the FIFO message buffer by reading the contents of that buffer. Once the buffer location is read, the application software clears the RXFUL bit corresponding to that buffer.

21.7.5 DeviceNet™ Filtering

The DeviceNet filtering feature is based on CAN 2.0A protocol 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 ECAN Control Register 2 (CiCTRL2<4:0>). The value specified in the DNCNT field determines the number of data bits to use for comparison with the EID bits of the message acceptance filter. If the CiCTRL2<DNCNT> bits 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 21-16, the first 18 data bits of the received message are compared with the corresponding identifier bits (EID<17:0>) of the message acceptance filter.

> **Note:** The DeviceNet filtering feature will only function when all of the following are true:
> - The message IDE bit = 0, which means the message is a standard ID message
> - The EXIDE bit in the CiRXFn register = 0
> - The MIDE bit in the CiRXMn register = 1
> - The value of the DNCNT bits is non-zero

Figure 21-16: ECAN™ Operation with DeviceNet™ Filtering

Note 1: The DeviceNet™ filtering configuration shown for EID bits is DNCNT<4:0> = 10010.
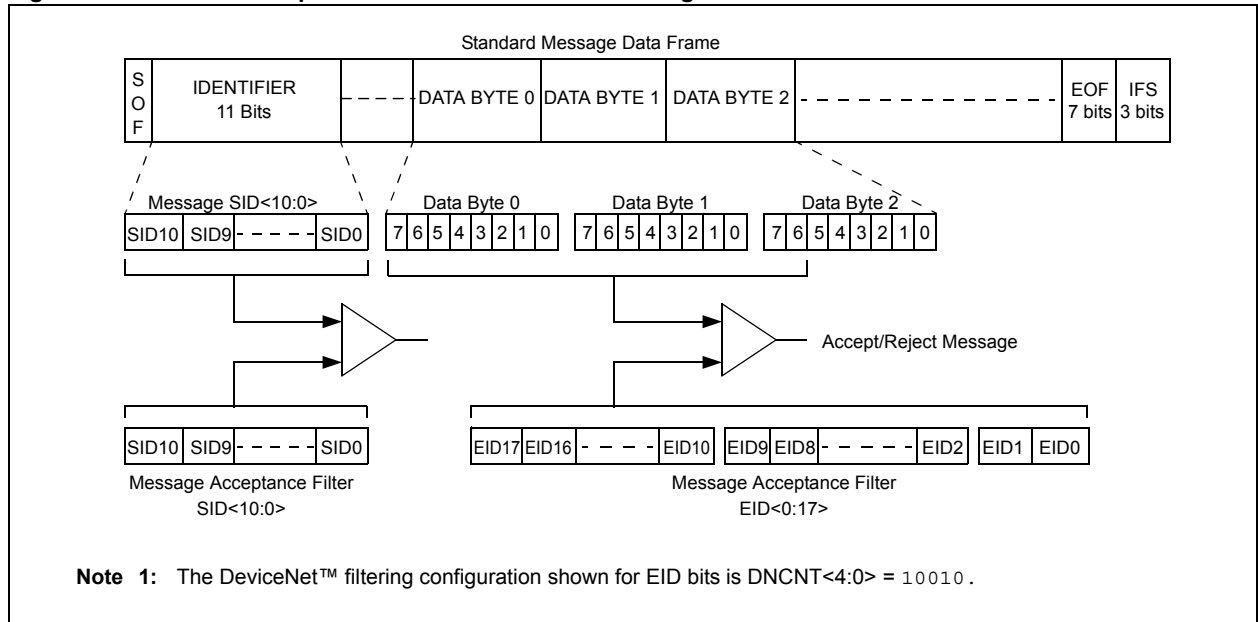
21.7.5.1 FILTER COMPARISONS

Table 21-4 shows the filter comparisons configured by the CiCTRL2(DNCNT<4:0>) control bits. For example, if DNCNT<4:0> = 00011, only a message in which the 11-bit Standard Identifier matches the SID acceptance filter (SID<10:0>) and bits 7, 6 and 5 of Data Byte 0 match the Extended Identifier filter (EID<0:2>) is accepted.

Table 21-4: 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<17>
00010	Data Byte 0<7:6>	EID<17:16>
00011	Data Byte 0<7:5>	EID<17:15>
00100	Data Byte 0<7:4>	EID<17:14>
00101	Data Byte 0<7:3>	EID<17:13>
00110	Data Byte 0<7:2>	EID<17:12>
00111	Data Byte 0<7:1>	EID<17:11>
01000	Data Byte 0<7:0>	EID<17:10>
01001	Data Byte 0<7:0> and Data Byte 1<7>	EID<17:9>
01010	Data Byte 0<7:0> and Data Byte 1<7:6>	EID<17:8>
01011	Data Byte 0<7:0> and Data Byte 1<7:5>	EID<17:7>
01100	Data Byte 0<7:0> and Data Byte 1<7:4>	EID<17:6>
01101	Data Byte 0<7:0> and Data Byte 1<7:3>	EID<17:5>
01110	Data Byte 0<7:0> and Data Byte 1<7:2>	EID<17:4>
01111	Data Byte 0<7:0> and Data Byte 1<7:1>	EID<17:3>
10000	Data Byte 0<7:0> and Data Byte 1<7:0>	EID<17:2>
10001	Byte 0<7:0> and Byte 1<7:0> and Byte 2<7>	EID<17:1>
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

21.7.5.2 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 18th 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 16th 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 8th 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 Standard Identifier. If the SID matches, the message is accepted.

21.8 DMA CONTROLLER CONFIGURATION

The ECAN module uses a portion of DMA RAM for message buffers to support both transmission and reception of CAN messages. The number of message buffers to be used by the ECAN module is specified by the DMA Buffer Size (DMABS<2:0>) bits in the ECAN FIFO Control (CiFCTRL<15:13>) register. The DMAxSTA register in the DMA controller defines the start of the CAN buffer area. The DMA controller moves data between ECAN and the DMA message buffers without CPU intervention.

The DMA controller provides eight channels for data transfer between DMA RAM and the PIC24H peripherals that are DMA ready. Two DMA channels are needed to support both CAN message transmission and reception. Each channel has a Request Register (DMAxREQ) to assign an event based on which message transfer occurs, as shown in Table 21-5.

Table 21-5: DMA Channel Request

Event	IRQ Number to Initialize DMAxREQ Register
ECAN1 Reception	34
ECAN1 Transmission	70
ECAN2 Reception	55
ECAN2 Transmission	71

An example showing the use of DMA Channel 4 to access the ECAN buffers is illustrated in Figure 21-17.

Figure 21-17: ECAN™ Message Buffer Memory Usage

Note: Refer to the specific device data sheet for information on the DMA base address.
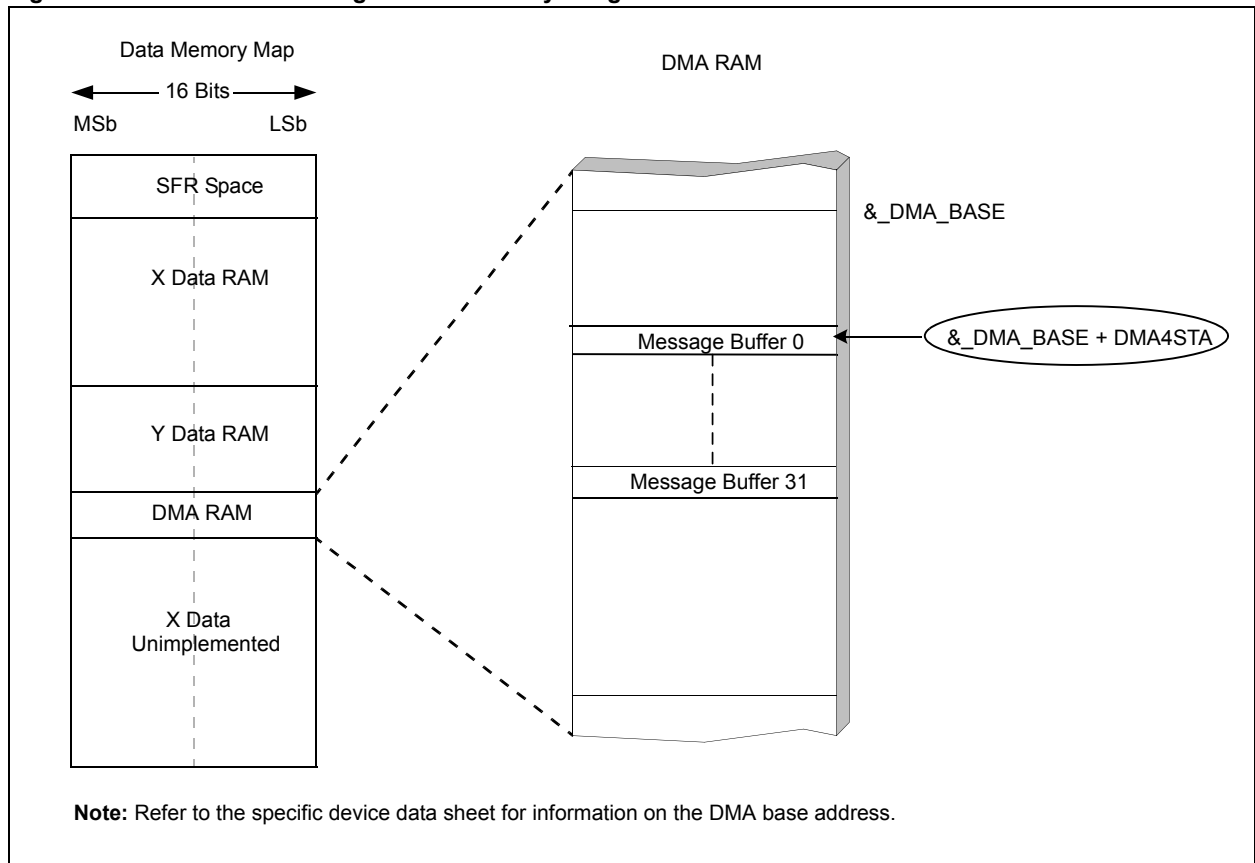
21.8.1 DMA Operation for Transmitting Data

The user application selects a message for transmission by setting the Message Send Request (TXREQ) bit in the ECAN TX/RX Buffer m Control (CiTRmnCON<3>) register. The ECAN controller uses DMA to read the message from the message buffer and transmit the message. The ECAN module generates a transmit data interrupt to start a DMA cycle. In response to the interrupt, the DMA channel that is configured for ECAN message transmission reads from the message buffer in DMA RAM and stores the message in the ECAN Transmit Data (CiTXD) register. Eight words are transferred for every message transmitted by the ECAN controller. **21.2 "CAN Message Formats"** provides the detailed layout of the transmit message buffer. Sample code for configuring the DMA channel for ECAN 1 transmission is shown in Example 21-6.

Example 21-6: DMA Channel 0 Configuration for ECAN1 Transmission

```
/* Data Transfer Size: Word Transfer Mode */
DMA0CONbits.SIZE = 0x0;

/* Data Transfer Direction: DMA RAM to Peripheral */
DMA0CONbits.DIR = 0x1;

/* DMA Addressing Mode: Peripheral Indirect Addressing mode */
DMA0CONbits.AMODE = 0x2;

/* Operating Mode: Continuous, Ping-Pong modes disabled */
DMA0CONbits.MODE = 0x0;

/* Assign ECAN1 Transmit event for DMA Channel 0 */
DMA0REQ = 70;

/* Set Number of DMA Transfer per ECAN message to 8 words */
DMA0CNT = 7;

/* Peripheral Address: ECAN1 Transmit Register */
DMA0PAD = &C1TXD;

/* Start Address Offset for ECAN1 Message Buffer 0x0000 */
DMA0STA = 0x0000;

/* Channel Enable: Enable DMA Channel 0 */
DMA0CONbits.CHEN = 0x1;

/* Channel Interrupt Enable: Enable DMA Channel 0 Interrupt */
IEC0bits DMA0IE = 1;
```

> **Note:** Please refer to **Section 22 "Direct Memory Access (DMA)"** (DS70223) for more information on how to configure the DMA Controller. Check for the most recent documentation on the Microchip web site at www.microchip.com.

21.8.2 DMA Operation for Receiving Data

When the ECAN controller completes a received message (8 words), the completed message is transferred to a message buffer in DMA RAM by the DMA. The ECAN module generates a receive data interrupt to start a DMA cycle. In response to this interrupt, the DMA channel that is configured for ECAN message reception reads from the ECAN Receive Data (CiRXD) register and stores the message in the DMA RAM buffer. Eight words are transferred for every message that is received by the ECAN controller. The detailed layout of the received message is provided in **21.2 "CAN Message Formats"**. Sample code for configuring the DMA channel for ECAN1 reception is shown in Example 21-7.

Example 21-7: DMA Channel 1 Configuration for ECAN1 Reception

```
/* Data Transfer Size: Word Transfer Mode */
DMA1CONbits.SIZE = 0x0;

/* Data Transfer Direction: Peripheral to DMA RAM */
DMA1CONbits.DIR = 0x0;

/* DMA Addressing Mode: Peripheral Indirect Addressing mode */
DMA1CONbits.AMODE = 0x2;

/* Operating Mode: Continuous, Ping-Pong modes disabled */
DMA1CONbits.MODE = 0x0;

/* Assign ECAN1 Receive event for DMA Channel 0 */
DMA1REQ = 34;

/* Set Number of DMA Transfer per ECAN message to 8 words */
DMA1CNT = 7;

/* Peripheral Address: ECAN1 Receive Register */
DMA1PAD = &C1RXD;

/* Start Address Offset for ECAN1 Message Buffer 0x0000 */
DMA1STA = 0x0000;

/* Channel Enable: Enable DMA Channel 1 */
DMA1CONbits.CHEN = 0x1;

/* Channel Interrupt Enable: Enable DMA Channel 1 Interrupt */
IEC0bits.DMA1IE = 1;
```

> **Note:** Please refer to **Section 22 "Direct Memory Access (DMA)"** (DS70223) for more information on how to configure the DMA Controller. Check for the most recent documentation on the Microchip web site at www.microchip.com.

21.9 BIT TIMING

The nominal bit rate is the number of bits per second transmitted on the CAN bus: Nominal Bit Time = 1 ÷ 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 (TQ). One TQ 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 and 25 TQ.

Figure 21-18 shows how the time quanta clock (F_{TQ}) is obtained from the system clock and also how the different time segments are programmed.

Figure 21-18: ECAN™ Bit Timing
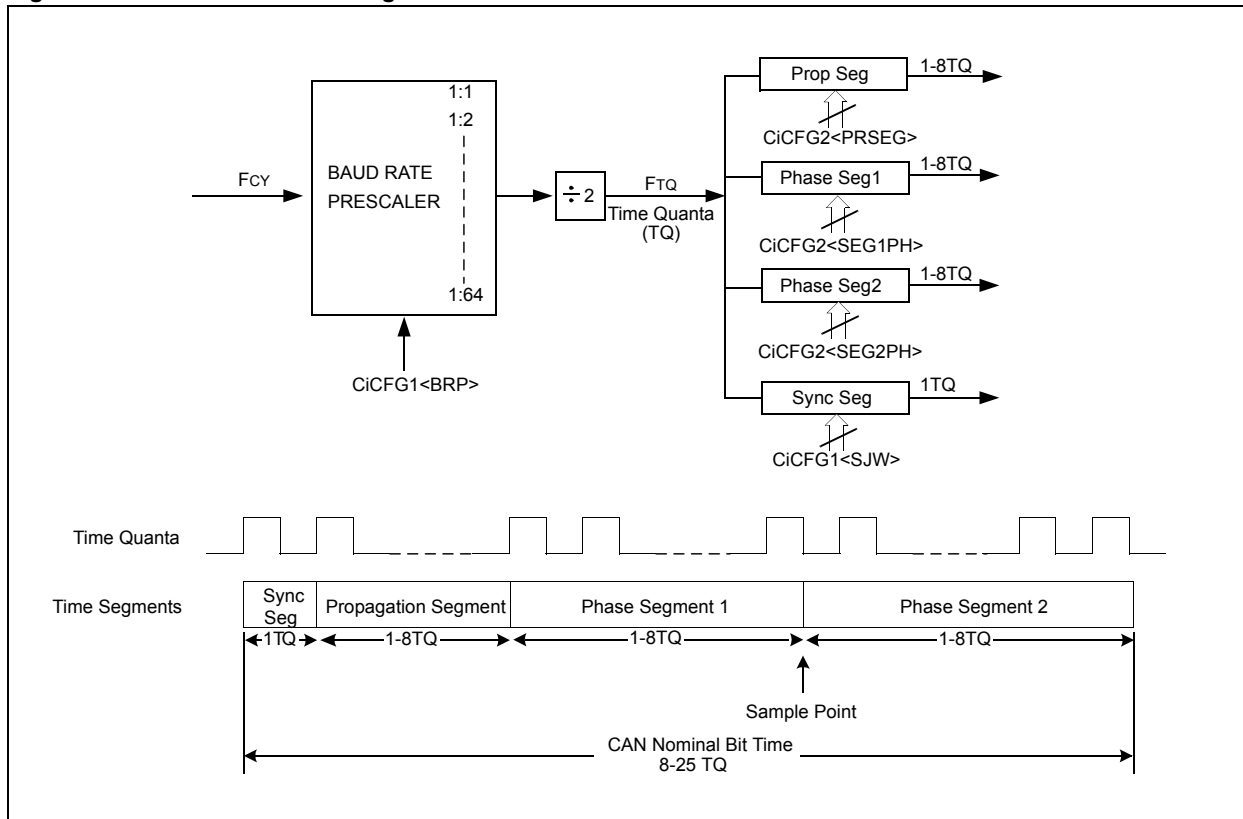
21.9.1 Bit Segments

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 one TQ.
- **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.

21.9.2   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 three times at the sample point, as configured by the Sample CAN bus Line (SAM) bit in ECAN Baud Rate Configuration Register (CiCFG2<6>).

• If CiCFG2<SAM> = 1, the CAN bus is sampled three times at the sample point. The most common of the three samples determines the bit value.

• If CiCFG2<SAM> = 0, the CAN bus is sampled only once at the sample point.

21.9.3   Synchronization

Two types of synchronization are used – Hard Synchronization and Resynchronization. A Hard Synchronization occurs once at the start of a frame. 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 Synchronization Jump Width parameter (CiCFG1<SJW>).

The length of Phase Segment 1 and Phase Segment 2 can be changed depending on oscillator tolerances of the transmitting 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 ECAN has a slower oscillator than the receiving node, the next falling edge, and hence the sample point, can be delayed by lengthening Phase Segment 1 in the bit time.

• **Bit Shortening** – If the transmitting node in ECAN has a faster oscillator than the receiving node, then the next falling edge, and hence 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 ECAN Baud Rate Configuration Register (CiCFG1<7:6>) determine the synchronization jump width 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 1-4 TQ.

21.9.4   ECAN Bit Time Calculations

The steps that need to be performed by the user application to configure the bit timing for the ECAN module are described below, which include examples.

1. Calculate the Time Quantum Frequency (F_{TQ}):
    a) Select the Baud Rate for the CAN Bus (F_{BAUD}).
    b) Select the number of tIme quanta in a bit time, based on your system requirements.

    The Time Quantum Frequency (F_{TQ}) is given by Equation 21-2:

Equation 21-2:   Time Quantum Frequency Calculation

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

> **Note 1:** The total number of time quanta in a nominal bit time must be programmed between 8 TQ and 25 TQ. Therefore the Time Quantum Frequency (F_{TQ}) is between 8 to 25 times baud rate (F_{BAUD}).
>
> **2:** Make sure that F_{TQ} is an integer multiple of F_{BAUD} to get precise bit time. If not, the oscillator input frequency or baud rate may need to be changed.

2. Calculate the Baud Rate Prescaler (CiCFG1<BRP>). The baud rate prescaler is given by Equation 21-3:

Equation 21-3:   Baud Rate Prescaler Calculation

$$CiCFG1(BRP) = \left(\frac{F_{CY}}{(2 \cdot F_{TQ})} \right) - 1$$

3. Select the Individual Bit Time Segments. The Individual Bit Time Segments are selected using the CiCFG2 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 Synchronous Jump Width.

Example 21-8:   CAN Bit Timing Calculation Example

- Step 1: Calculate the time quantum frequency (F_{CY} = 40 MHz).
  If F_{BAUD} = 1 Mbps, and number of time quanta N = 20, then F_{TQ} = 20 MHz.
- Step 2: Calculate the baud rate prescaler:
  CiCFG1<BRP> = (40000000/(2 * F_{TQ})) - 1 = 1 - 1 = 0.
- Step 3: Select the individual bit time segments.
  - Synchronization Segment = 1TQ (constant).
  - Based on system characteristics, suppose the Propagation Delay = 5 TQ.
  - Suppose the sample point is to be at 70% of Nominal Bit Time. Phase Segment 2 = 30% of Nominal Bit Time = 6 TQ.
  - Then, Phase Segment 1 = 20 TQ - (1 TQ + 5 TQ + 6 TQ) = 8 TQ.

Example 21-9 illustrates code for configuring ECAN bit timing parameters.

Example 21-9:   Code Example for Configuring ECAN™ Bit Timing Parameters

```
/* Set the Operating Frequency of the device to be 40MHz */

#define FCY 40000000

/* Set the ECAN module for Configuration Mode before writing into the Baud
Rate Control Registers*/

C1CTRL1bits.REQOP = 4;

/* Wait for the ECAN module to enter into Configuration Mode */

while(C1CTRL1bits.OPMODE! = 4);

/* Phase Segment 1 time is 8 TQ */
C1CFG2bits.SEG1PH = 0x7;

/* Phase Segment 2 time is set to be programmable */

C1CFG2bits.SEG2PHTS = 0x1;

/* Phase Segment 2 time is 6 TQ */

C1CFG2bits.SEG2PH = 0x5;

/* Propagation Segment time is 5 TQ */

C1CFG2bits.PRSEG = 0x4;

/* Bus line is sampled three times at the sample point */

C1CFG2bits.SAM = 0x1;

/* Synchronization Jump Width set to 4 TQ */

C1CFG1bits.SJW = 0x3;

/* Baud Rate Prescaler bits set to 1:1, i.e., TQ = (2*1*1)/ FCAN */

C1CFG1bits.BRP = 0x0 ;

/* Put the ECAN Module into Normal Mode Operating Mode*/

C1CTRL1bits.REQOP = 0;

/* Wait for the ECAN module to enter into Normal Operating Mode */

while(C1CTRL1bits.OPMODE! = 0);
```

21.10 ECAN ERROR MANAGEMENT

21.10.1 CAN Bus Errors

The CAN protocol 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.

21.10.1.1 BIT ERROR

A node that is sending a bit on the bus also monitors the bus. A bit error is detected when the monitored bit value is different from the sent bit value. 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.

21.10.1.2 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.

21.10.1.3 FORM ERROR

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

21.10.1.4 STUFFING ERROR

A stuffing error is detected at the bit time of the 6th consecutive equal bit level in a message field that should be coded by the method of bit stuffing.

21.10.1.5 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.

21.10.2 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 (i.e., discard the current message).

The ECAN module maintains two error counters:

• Transmit Error Counter (CiEC<TERRCNT>)
• Receive Error Counter (CiEC<RERRCNT>)

There are several rules governing how these counters are incremented and/or decremented. In essence, a transmitter detecting a fault increments its transmit error counter faster than the listening nodes can 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 CAN 2.0B specification.

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 Bus Off transmits nothing on the bus.

In addition, the ECAN 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 21-19.

Figure 21-19:   Error Modes
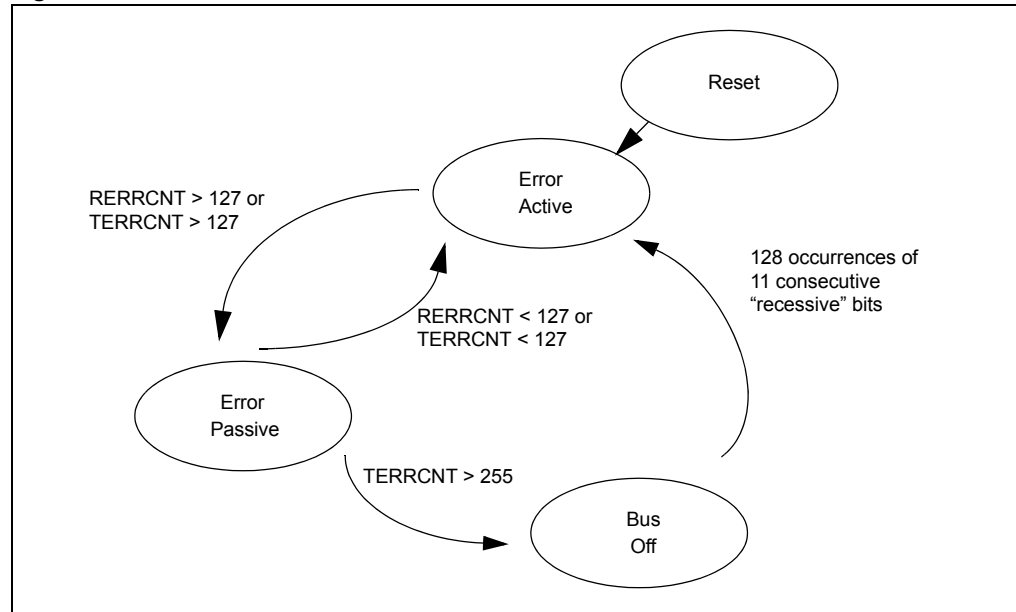
21.10.2.1 TRANSMITTER IN ERROR PASSIVE STATE

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

21.10.2.2 RECEIVER IN ERROR PASSIVE STATE

The Receiver Error Passive (CiINTF<RXBP>) bit is set when the Receive Error Counter equals or exceeds 128 and generates an error interrupt (CiINTF<ERRIF>) 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.

21.10.2.3 TRANSMITTER IN BUS OFF STATE

The Transmitter Bus Off (CiINTF<TXBO>) bit is set when the Transmit Error Counter equals or exceeds 256 and generates an error interrupt (CiINTF<ERRIF>).

21.10.2.4 TRANSMITTER IN ERROR WARNING STATE

The Transmitter Error Warn (CiINTF<TXWAR>) bit is set when the Transmit Error Counter is in the range of 96 and 127 (inclusive) and generates an error interrupt (CiINTF<ERRIF>) upon entry into Error Warn state. The Transmit Error Warn flag is cleared automatically by the hardware if the Transmit Error Counter becomes less than 96 or more than 127 (i.e., the ECAN module has entered a bus passive state).

21.10.2.5 RECEIVER IN ERROR WARNING STATE

The Receiver Error Warn (CiINTF<RXWAR>) bit is set when the Receive Error Counter is in the range of 96 and 127 (inclusive) and generates an error interrupt (CiINTF<ERRIF>) upon entry into Error Warn state. The Receive Error Warn flag is cleared automatically by the hardware if the Receive Error Counter becomes less than 96 or more than 127 (i.e., the ECAN module has entered a bus passive state).

Additionally, there is an Error State Warning flag (CiINTF<EWARN>) bit, 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.

21.11   ECAN INTERRUPTS

ECAN module generates three different interrupts, each with its own interrupt vector, interrupt enable control bit, interrupt status flag and interrupt priority control bit. These interrupts are:

- **CiTX** – ECAN Transmit Data Request Interrupt
- **CiRX** – ECAN Receive Data Ready Interrupt
- **Ci** – ECAN Event Interrupt

21.11.1   ECAN Transmit Data Request Interrupt

The transmit data request interrupt represents the transmission of a single word in an ECAN message through the ECAN Transmit Data (CiTXD) register. The user application needs to assign the ECAN transmit data request interrupt to a DMA channel to automatically transfer messages from the appropriate DMA RAM buffers to the ECAN module (CiTXD register).

21.11.2   ECAN Receive Data Ready Interrupt

The receive data request interrupt represents the reception of a single word of an ECAN message through the ECAN Receive Data (CiRXD) register. The user application needs to assign the ECAN receive data ready interrupt to a DMA channel to automatically transfer messages from the ECAN module (CiRXD register) to the appropriate DMA RAM buffers.

21.11.3   ECAN Event Interrupt

The ECAN event interrupt has seven main sources, each of which can be individually enabled. The Interrupt Flag (CiINTF) register contains the interrupt flags, and the Interrupt Enable (CiINTE) register contains the enable bits. The Interrupt Flag Code bits (ICODE<6:0>) in the ECAN Interrupt Code (CiVEC<6:0>) register can be used in combination with a jump table for efficient handling of interrupts. All interrupts have one source, with the exception of the Error Interrupt. Any of five error interrupt sources (TX Error Warn, RX Error Warn, TX Error Passive, RX Error Passive and TX Bus Off) can set an error interrupt flag. The source of the error interrupt is determined by reading the CiINTF register. The ECAN module will generate an ECAN event interrupt (Ci interrupt) only if, atleast one of the conditions in the CiINTF register is active, and the corresponding interrupt is enabled in the CiINTE register. The CPU will vector to the CAN event interrupt service routine, if CiIF bit and the CiIE bit are set..

> **Note:**   The ICODE<6:0> bits will reflect the highest priority CAN interrupt condition that is active. The interrupt should be enabled (the IE bit in the CiINTE register should be set) and the interrupt condition should be active (the IF bit in the CiINTF register should be set).

Figure 21-20 shows ECAN event interrupt generation from various interrupt sources.

21.11.3.1   TRANSMIT BUFFER INTERRUPT

The message buffers 0 to 7 that are configured for message transmission generate the transmit buffer interrupt (CiINTF<TBIF>) bit after the CAN message is transmitted. The ICODE bits indicate the specific message buffer that generated the transmit buffer interrupt. Transmit buffer interrupts must be cleared in the Interrupt Service Routine by clearing the TBIF bit.

21.11.3.2   RECEIVE BUFFER INTERRUPT

When a message is successfully received and loaded into one of the receive buffers (message buffers 0 to 31), the receive buffer interrupt (CiINTF<RBIF>) is activated after the module sets the CiRXFULm<RXFULn> bit. The ICODE bits indicates the particular buffer which generated interrupt. The receive buffer interrupt must be cleared in the Interrupt Service Routine by clearing RBIF bit.

21.11.3.3 RECEIVE BUFFER OVERFLOW INTERRUPT

When a message is successfully received but the designated buffer is full, the receive overflow interrupt (CiINTF<RBOVIF>) is activated after the module sets the CiRXOVFm<RXOVFn> bit. The Interrupt Flag Code (ICODE<6:0>) bits in the ECAN Interrupt Code (CiVEC<6:0>) register indicates which buffer generated the interrupt. The receive buffer overflow interrupt must be cleared in the Interrupt Service Routine by clearing RBOVIF bit.

21.11.3.4 FIFO ALMOST FULL INTERRUPT

When the FIFO has only one remaining available buffer, the FIFO interrupt (CiINTF<FIFOIF>) is activated after the module sets the CiRXFULm<RXFULn> bit for the next to last available buffer. The CiVEC<ICODE> bits indicate the FIFO overflow condition. The FIFO almost full interrupt must be cleared in the Interrupt Service Routine by clearing the CiINTF<FIFOIF> bit.

21.11.3.5 ERROR INTERRUPT

The error interrupt (CiINTF<ERRIF>) is generated by five sources:

- TX Error Warn
- RX Error Warn
- TX Error Passive
- RX Error Passive
- TX Bus Off

The CiVEC<ICODE> bits indicate the Error condition. The error interrupt must be cleared in the Interrupt Service Routine by clearing the CiINTF<ERRIF> bit.

21.11.3.6 WAKE-UP INTERRUPT

In Sleep mode, the device monitors the ECAN receive pin (CiRX) for bus activity. A wake-up (CiINTF<WAKIF>) interrupt is generated when bus activity is detected. The CiVEC<ICODE> bits indicate the Wake-up condition. The wake-up interrupt must be cleared in the Interrupt Service Routine by clearing the CiINTF<WAKIF> bit.

21.11.3.7 INVALID MESSAGE RECEIVED INTERRUPT

The invalid message received interrupt is generated for any other type of errors during message reception.

Section 21. Enhanced Controller Area Network (ECAN™)

Figure 21-20: ECAN™ Event Interrupts
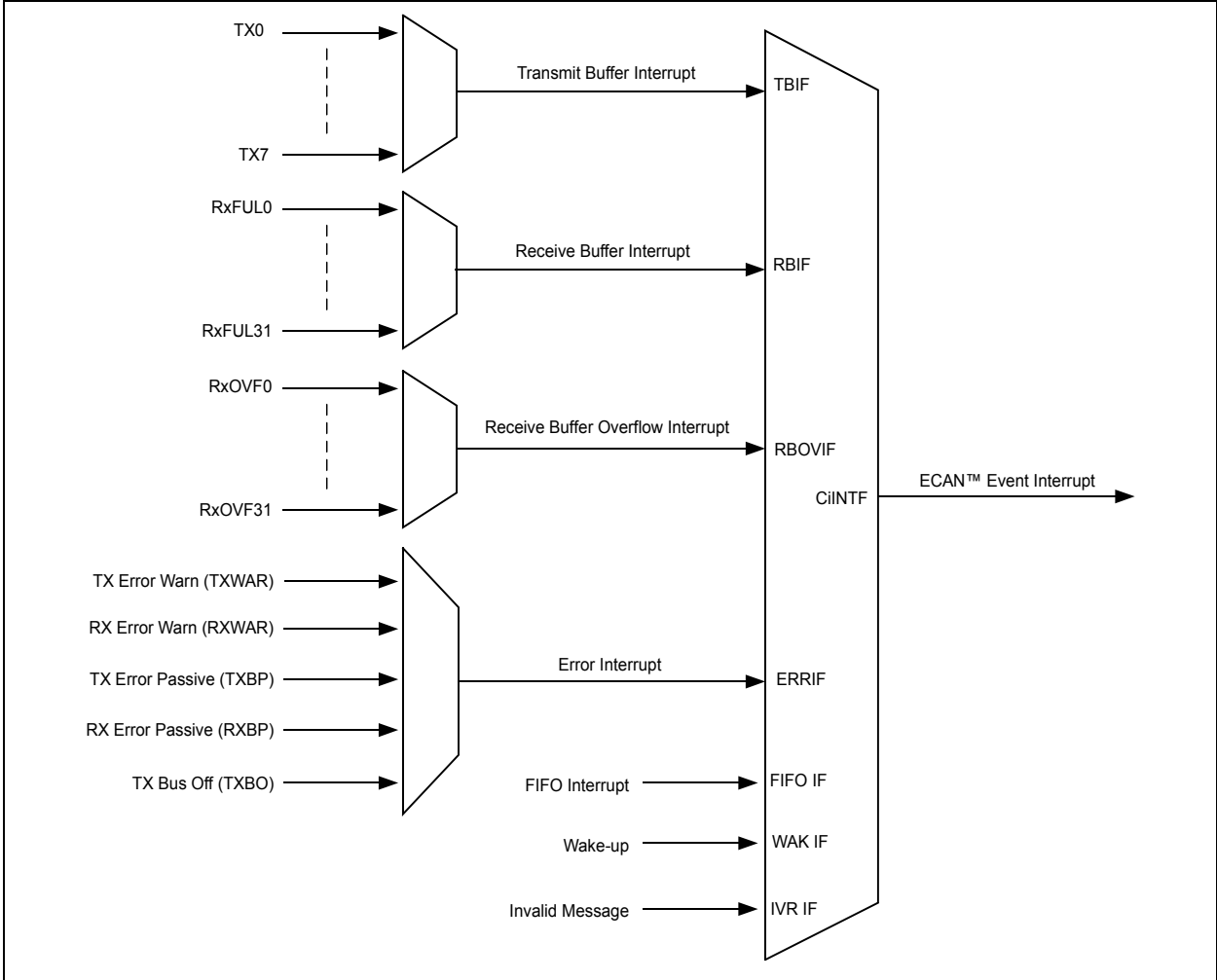
21.12 ECAN LOW-POWER MODES

The ECAN module can respond to the CPU `PWRSAV` instruction.

21.12.1 Sleep Mode

A CPU `PWRSAV,1` instruction stops the crystal oscillator and shuts down all system clocks. The user application must ensure that the 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 module drives the CiTX pin into the recessive state while sleeping. The recommended procedure is to bring the module into Disable mode before executing the CPU `PWRSAV` instruction.

21.12.2 Idle Mode

A CPU `PWRSAV,0` instruction signals the module to optionally shut down clocks. The module powers down if the Stop in Idle Mode (CSIDL) bit in ECAN Control Register 1 (CiCTRL1<13>) is '1'. The user application must ensure that the module is not active when the CPU goes into Idle mode. To protect the CAN bus system from fatal consequences due to violations of the above rule, the module drives the CiTX pin into the recessive state while sleeping. The recommended procedure is to bring the module into Disable mode before executing the CPU `PWRSAV` instruction.

21.12.3 Wake-up Functions

The module monitors the RX line for activity while the device is sleeping. If the WAKIE bit is set, the module generates an interrupt if bus activity is detected. Due to the delays in starting up the oscillator and CPU, the message activity that caused the wake-up is lost.

After the CPU wakes up from Sleep, the CPU executes the CAN event Interrupt Service Routine (if interrupts are enabled); however, the CAN module itself would still be disabled. The CAN bus wake-up feature only works when the device is in Sleep mode.

The module features a low-pass filter on the CiRX input line, which should be enabled when the module is in CPU Sleep mode. This filter protects the module from wake-up due to short glitches on the CAN bus. the filter is enabled by setting the CiCFG2<WAKFIL> bit.

21.13 ECAN TIME STAMPING USING INPUT CAPTURE

The ECAN module generates a signal that can be sent to a timer capture input whenever a valid frame is accepted. This is useful for time-stamping and network synchronization. Because the CAN specification defines a frame to be valid if no errors occurred before the EOF field has been transmitted successfully, the timer signal is generated immediately after the EOF. A pulse of one bit time is generated.

Time-stamping is enabled by the CAN Message Receive Timer Capture Event Enable (CANCAP) control bit (CiCTRL1<3>). The IC2 capture input is used for time-stamping.

> **Note:** If the CAN capture is enabled, the IC2 pin becomes unusable as a general input capture pin. In this mode, the IC2 channel derives its input signal from the C1RX or C2RX pin instead of the IC2 pin.

21.14 REGISTER MAPS

The following are the register maps related to ECAN:
- ECAN1 Register Map When C1CTRL1.WIN = 0 or 1 (see Table 21-6)
- ECAN1 Register Map When C1CTRL1.WIN = 0 (see Table 21-7)
- ECAN1 Register Map When C1CTRL1.WIN = 1 (see Table 21-8)
- ECAN2 Register Map When C2CTRL1.WIN = 0 or 1 (see Table 21-9)
- ECAN2 Register Map When C2CTRL1.WIN = 0 (see Table 21-10)
- ECAN2 Register Map When C2CTRL1.WIN = 1 (see Table 21-11)

Table 21-6: **ECAN1 Register Map When C1CTRL1.WIN = 0 or 1**

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
C1CTRL1	—	—	CSIDL	ABAT	—	REQOP<2:0>			OPMODE<2:0>			—	CANCAP	—	—	WIN	0480
C1CTRL2	—	—	—	—	—	—	—	—	—	—	—	DNCNT<4:0>					0000
C1VEC	—	—	—	FILHIT<4:0>					—	ICODE<6:0>							0040
C1FCTRL	DMABS<2:0>			—	—	—	—	—	—	—	—	FSA<4:0>					0000
C1FIFO	—	—	FBP<5:0>						—	—	FNRB<5:0>						0000
C1INTF	—	—	TXBO	TXBP	RXBP	TXWAR	RXWAR	EWARN	IVRIF	WAKIF	ERRIF	—	FIFOIF	RBOVIF	RBIF	TBIF	0000
C1INTE	—	—	—	—	—	—	—	—	IVRIE	WAKIE	ERRIE	—	FIFOIE	RBOVIE	RBIE	TBIE	0000
C1EC	TERRCNT<7:0>								RERRCNT<7:0>								0000
C1CFG1	—	—	—	—	—	—	—	—	SJW<1:0>		BRP<5:0>						0000
C1CFG2	—	WAKFIL	—	—	—	SEG2PH<2:0>			SEG2PHTS	SAM	SEG1PH<2:0>			PRSEG<2:0>			0000
C1FEN1	FLTEN15	FLTEN14	FLTEN13	FLTEN12	FLTEN11	FLTEN10	FLTEN9	FLTEN8	FLTEN7	FLTEN6	FLTEN5	FLTEN4	FLTEN3	FLTEN2	FLTEN1	FLTEN0	0000
C1FMSKSEL1	F7MSK<1:0>		F6MSK<1:0>		F5MSK<1:0>		F4MSK<1:0>		F3MSK<1:0>		F2MSK<1:0>		F1MSK<1:0>		F0MSK<1:0>		0000
C1FMSKSEL2	F15MSK<1:0>		F14MSK<1:0>		F13MSK<1:0>		F12MSK<1:0>		F11MSK<1:0>		F10MSK<1:0>		F9MSK<1:0>		F8MSK<1:0>		0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-7: **ECAN1 Register Map When C1CTRL1.WIN = 0**

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
								See definition when WIN = x									
C1RXFUL1	RXFUL15	RXFUL14	RXFUL13	RXFUL12	RXFUL11	RXFUL10	RXFUL9	RXFUL8	RXFUL7	RXFUL6	RXFUL5	RXFUL4	RXFUL3	RXFUL2	RXFUL1	RXFUL0	0000
C1RXFUL2	RXFUL31	RXFUL30	RXFUL29	RXFUL28	RXFUL27	RXFUL26	RXFUL25	RXFUL24	RXFUL23	RXFUL22	RXFUL21	RXFUL20	RXFUL19	RXFUL18	RXFUL17	RXFUL16	0000
C1RXOVF1	RXOVF15	RXOVF14	RXOVF13	RXOVF12	RXOVF11	RXOVF10	RXOVF9	RXOVF8	RXOVF7	RXOVF6	RXOVF5	RXOVF4	RXOVF3	RXOVF2	RXOVF1	RXOVF0	0000
C1RXOVF2	RXOVF31	RXOVF30	RXOVF29	RXOVF28	RXOVF27	RXOVF26	RXOVF25	RXOVF24	RXOVF23	RXOVF22	RXOVF21	RXOVF20	RXOVF19	RXOVF18	RXOVF17	RXOVF16	0000
C1TR01CON	TXEN1	TXABT1	TXLARB1	TXERR1	TXREQ1	RTREN1	TX1PRI<1:0>		TXEN0	TXABT0	TXLARB0	TXERR0	TXREQ0	RTREN0	TX0PRI<1:0>		0000
C1TR23CON	TXEN3	TXABT3	TXLARB3	TXERR3	TXREQ3	RTREN3	TX3PRI<1:0>		TXEN2	TXABT2	TXLARB2	TXERR2	TXREQ2	RTREN2	TX2PRI<1:0>		0000
C1TR45CON	TXEN5	TXABT5	TXLARB5	TXERR5	TXREQ5	RTREN5	TX5PRI<1:0>		TXEN4	TXABT4	TXLARB4	TXERR4	TXREQ4	RTREN4	TX4PRI<1:0>		0000
C1TR67CON	TXEN7	TXABT7	TXLARB7	TXERR7	TXREQ7	RTREN7	TX7PRI<1:0>		TXEN6	TXABT6	TXLARB6	TXERR6	TXREQ6	RTREN6	TX6PRI<1:0>		xxxx
C1RXD	Received Data Word																xxxx
C1TXD	Transmit Data Word																xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

PIC24H Family Reference Manual

Table 21-8: ECAN1 Register Map When C1CTRL1.WIN = 1

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
							See definition when WIN = x										
C1BUFPNT1	F3BP<3:0>				F2BP<3:0>				F1BP<3:0>				F0BP<3:0>				0000
C1BUFPNT2	F7BP<3:0>				F6BP<3:0>				F5BP<3:0>				F4BP<3:0>				0000
C1BUFPNT3	F11BP<3:0>				F10BP<3:0>				F9BP<3:0>				F8BP<3:0>				0000
C1BUFPNT4	F15BP<3:0>				F14BP<3:0>				F13BP<3:0>				F12BP<3:0>				0000
C1RXM0SID	SID<10:3>								SID<2:0>			—	MIDE	—	EID<17:16>		xxxx
C1RXM0EID	EID<15:8>								EID<7:0>								xxxx
C1RXM1SID	SID<10:3>								SID<2:0>			—	MIDE	—	EID<17:16>		xxxx
C1RXM1EID	EID<15:8>								EID<7:0>								xxxx
C1RXM2SID	SID<10:3>								SID<2:0>			—	MIDE	—	EID<17:16>		xxxx
C1RXM2EID	EID<15:8>								EID<7:0>								xxxx
C1RXF0SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF0EID	EID<15:8>								EID<7:0>								xxxx
C1RXF1SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF1EID	EID<15:8>								EID<7:0>								xxxx
C1RXF2SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF2EID	EID<15:8>								EID<7:0>								xxxx
C1RXF3SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF3EID	EID<15:8>								EID<7:0>								xxxx
C1RXF4SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF4EID	EID<15:8>								EID<7:0>								xxxx
C1RXF5SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF5EID	EID<15:8>								EID<7:0>								xxxx
C1RXF6SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF6EID	EID<15:8>								EID<7:0>								xxxx
C1RXF7SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF7EID	EID<15:8>								EID<7:0>								xxxx
C1RXF8SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF8EID	EID<15:8>								EID<7:0>								xxxx
C1RXF9SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF9EID	EID<15:8>								EID<7:0>								xxxx
C1RXF10SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF10EID	EID<15:8>								EID<7:0>								xxxx
C1RXF11SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-8: **ECAN1 Register Map When C1CTRL1.WIN = 1 (Continued)**

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
C1RXF11EID	EID<15:8>								EID<7:0>								xxxx
C1RXF12SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF12EID	EID<15:8>								EID<7:0>								xxxx
C1RXF13SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF13EID	EID<15:8>								EID<7:0>								xxxx
C1RXF14SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF14EID	EID<15:8>								EID<7:0>								xxxx
C1RXF15SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C1RXF15EID	EID<15:8>								EID<7:0>								xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-9: ECAN2 Register Map When C2CTRL1.WIN = 0 or 1

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
C2CTRL1	—	—	CSIDL	ABAT	—	REQOP<2:0>			OPMODE<2:0>			—	CANCAP	—	—	WIN	0480
C2CTRL2	—	—	—	—	—	—	—	—	—	—	—	DNCNT<4:0>					0000
C2VEC	—	—	—	FILHIT<4:0>					—	ICODE<6:0>							0040
C2FCTRL	DMABS<2:0>			—	—	—	—	—	—	—	—	FSA<4:0>					0000
C2FIFO	—	—	FBP<5:0>						—	—	FNRB<5:0>						0000
C2INTF	—	—	TXBO	TXBP	RXBP	TXWAR	RXWAR	EWARN	IVRIF	WAKIF	ERRIF	—	FIFOIF	RBOVIF	RBIF	TBIF	0000
C2INTE	—	—	—	—	—	—	—	—	IVRIE	WAKIE	ERRIE	—	FIFOIE	RBOVIE	RBIE	TBIE	0000
C2EC	TERRCNT<7:0>								RERRCNT<7:0>								0000
C2CFG1	—	—	—	—	—	—	—	—	SJW<1:0>		BRP<5:0>						0000
C2CFG2	—	WAKFIL	—	—	—	SEG2PH<2:0>			SEG2PHTS	SAM	SEG1PH<2:0>			PRSEG<2:0>			0000
C2FEN1	FLTEN15	FLTEN14	FLTEN13	FLTEN12	FLTEN11	FLTEN10	FLTEN9	FLTEN8	FLTEN7	FLTEN6	FLTEN5	FLTEN4	FLTEN3	FLTEN2	FLTEN1	FLTEN0	0000
C2FMSKSEL1	F7MSK<1:0>		F6MSK<1:0>		F5MSK<1:0>		F4MSK<1:0>		F3MSK<1:0>		F2MSK<1:0>		F1MSK<1:0>		F0MSK<1:0>		0000
C2FMSKSEL2	F15MSK<1:0>		F14MSK<1:0>		F13MSK<1:0>		F12MSK<1:0>		F11MSK<1:0>		F10MSK<1:0>		F9MSK<1:0>		F8MSK<1:0>		0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-10: ECAN2 Register Map When C2CTRL1.WIN = 0

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
						See definition when WIN = x											
C2RXFUL1	RXFUL15	RXFUL14	RXFUL13	RXFUL12	RXFUL11	RXFUL10	RXFUL9	RXFUL8	RXFUL7	RXFUL6	RXFUL5	RXFUL4	RXFUL3	RXFUL2	RXFUL1	RXFUL0	0000
C2RXFUL2	RXFUL31	RXFUL30	RXFUL29	RXFUL28	RXFUL27	RXFUL26	RXFUL25	RXFUL24	RXFUL23	RXFUL22	RXFUL21	RXFUL20	RXFUL19	RXFUL18	RXFUL17	RXFUL16	0000
C2RXOVF1	RXOVF15	RXOVF14	RXOVF13	RXOVF12	RXOVF11	RXOVF10	RXOVF09	RXOVF08	RXOVF7	RXOVF6	RXOVF5	RXOVF4	RXOVF3	RXOVF2	RXOVF1	RXOVF0	0000
C2RXOVF2	RXOVF31	RXOVF30	RXOVF29	RXOVF28	RXOVF27	RXOVF26	RXOVF25	RXOVF24	RXOVF23	RXOVF22	RXOVF21	RXOVF20	RXOVF19	RXOVF18	RXOVF17	RXOVF16	0000
C2TR01CON	TXEN1	TXABAT1	TXLARB1	TXERR1	TXREQ1	RTREN1	TX1PRI<1:0>		TXEN0	TXABAT0	TXLARB0	TXERR0	TXREQ0	RTREN0	TX0PRI<1:0>		0000
C2TR23CON	TXEN3	TXABAT3	TXLARB3	TXERR3	TXREQ3	RTREN3	TX3PRI<1:0>		TXEN2	TXABAT2	TXLARB2	TXERR2	TXREQ2	RTREN2	TX2PRI<1:0>		0000
C2TR45CON	TXEN5	TXABAT5	TXLARB5	TXERR5	TXREQ5	RTREN5	TX5PRI<1:0>		TXEN4	TXABAT4	TXLARB4	TXERR4	TXREQ4	RTREN4	TX4PRI<1:0>		0000
C2TR67CON	TXEN7	TXABAT7	TXLARB7	TXERR7	TXREQ7	RTREN7	TX7PRI<1:0>		TXEN6	TXABAT6	TXLARB6	TXERR6	TXREQ6	RTREN6	TX6PRI<1:0>		xxxx
C2RXD	Recieved Data Word																xxxx
C2TXD	Transmit Data Word																xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-11: ECAN2 Register Map When C2CTRL1.WIN = 1

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
								See definition when WIN = x									
C2BUFPNT1		F3BP<3:0>				F2BP<3:0>				F1BP<3:0>				F0BP<3:0>			0000
C2BUFPNT2		F7BP<3:0>				F6BP<3:0>				F5BP<3:0>				F4BP<3:0>			0000
C2BUFPNT3		F11BP<3:0>				F10BP<3:0>				F9BP<3:0>				F8BP<3:0>			0000
C2BUFPNT4		F15BP<3:0>				F14BP<3:0>				F13BP<3:0>				F12BP<3:0>			0000
C2RXM0SID					SID<10:3>						SID<2:0>		—	MIDE	—	EID<17:16>	xxxx
C2RXM0EID					EID<15:8>								EID<7:0>				xxxx
C2RXM1SID					SID<10:3>						SID<2:0>		—	MIDE	—	EID<17:16>	xxxx
C2RXM1EID					EID<15:8>								EID<7:0>				xxxx
C2RXM2SID					SID<10:3>						SID<2:0>		—	MIDE	—	EID<17:16>	xxxx
C2RXM2EID					EID<15:8>								EID<7:0>				xxxx
C2RXF0SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF0EID					EID<15:8>								EID<7:0>				xxxx
C2RXF1SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF1EID					EID<15:8>								EID<7:0>				xxxx
C2RXF2SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF2EID					EID<15:8>								EID<7:0>				xxxx
C2RXF3SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF3EID					EID<15:8>								EID<7:0>				xxxx
C2RXF4SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF4EID					EID<15:8>								EID<7:0>				xxxx
C2RXF5SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF5EID					EID<15:8>								EID<7:0>				xxxx
C2RXF6SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF6EID					EID<15:8>								EID<7:0>				xxxx
C2RXF7SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF7EID					EID<15:8>								EID<7:0>				xxxx
C2RXF8SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF8EID					EID<15:8>								EID<7:0>				xxxx
C2RXF9SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF9EID					EID<15:8>								EID<7:0>				xxxx
C2RXF10SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx
C2RXF10EID					EID<15:8>								EID<7:0>				xxxx
C2RXF11SID					SID<10:3>						SID<2:0>		—	EXIDE	—	EID<17:16>	xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Table 21-11:   ECAN2 Register Map When C2CTRL1.WIN = 1 (Continued)

File Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
C2RXF11EID	EID<15:8>								EID<7:0>								xxxx
C2RXF12SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C2RXF12EID	EID<15:8>								EID<7:0>								xxxx
C2RXF13SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C2RXF13EID	EID<15:8>								EID<7:0>								xxxx
C2RXF14SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C2RXF14EID	EID<15:8>								EID<7:0>								xxxx
C2RXF15SID	SID<10:3>								SID<2:0>			—	EXIDE	—	EID<17:16>		xxxx
C2RXF15EID	EID<15:8>								EID<7:0>								xxxx

Legend: x = unknown value on Reset, — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

21.15   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 PIC24H product family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Enhanced Controller Area Network (ECAN™) module are:

Title	Application Note #
No application notes at this time.	N/A

> **Note:**   Please visit the Microchip web site (www.microchip.com) for additional Application Notes and code examples for the PIC24H family of devices.

21.16   REVISION HISTORY

Revision A (February 2007)

This is the initial released version of this document.

Revision B (April 2007)

Minor changes were made to this document.

Revision C (November 2009)

This revision includes the following updates:

- **Global Change:**
  - Removed all references to the CANCKS bit in Figure 21-18, **21.9.4 "ECAN Bit Time Calculations"**, Example 21-8, Example 21-9, Table 21-6 and Table 21-9.
- **Examples:**
  - Updated Example 21-1: Code Example for Standard Data Frame Transmission to reflect the maximum CAN message size of 8 (0x0008). The example incorrectly stated the value of 15 (0x000F).
  - Updated the code in Example 21-2: Code Example for Extended Data Frame Transmission to reflect message buffer 2 instead of message buffer 0
- **Figures:**
  - Updated the Logical value of SRR bit from '0' to '1' in Figure 21-5
- **Notes:**
  - Added a shaded note to **21.7.2.2 "Receiving Messages into Message Buffers 0-7"**, which details the effects of setting the TXREQ bit when the buffer is configured for receive operation
  - Added a shaded note to **21.7.5 "DeviceNet™ Filtering"**, which details operation constraints
  - Added a shaded note to **21.11.3 "ECAN Event Interrupt"**, which provides additional details on the highest priority CAN interrupt condition
- **Registers:**
  - Removed bit 11 (CANCKS) from CiFCTRL: ECAN FIFO Control Register (see Register 21-18).
  - Changed the default value of R/W-0 to R/W-1 for FLTEN6-FLTEN15 in CiFEN1: ECAN Acceptance Filter Enable Register (see Register 21-3)
  - Added shaded note to CiRXFnSID: ECAN Acceptance Filter Standard Identifier Register (see Register 21-4)
  - F7MSK<1:0> = 11 was changed from "No Mask" to "Reserved" in CiFMSKSEL1: ECAN Filter 7-0 Mask Selection Register (see Register 21-8)
  - F15MSK<1:0> = 11 was changed from "No Mask" to "Reserved" in CiFMSKSEL2: ECAN Filter 15-8 Mask Selection Register (see Register 21-9)
- **Sections:**
  - Updated the Logical value of the SRR bit from '0' to '1' in **21.2.2 "Extended Data Frame"**
  - Removed the references to the WIN bit in **21.3.6 "ECAN Control and Error Counter Registers"**
  - Changed "CiERR" to "CiEC" in **21.10.2 "Fault Confinement"**
  - Updated the Transmit Error Counter range in **21.10.2.4 "Transmitter in error warning State"**
  - Updated the Receive Error Counter range in **21.10.2.5 "Receiver in error warning state"**
  - Changed TXCAN to CiTX in **21.12.1 "Sleep Mode"**
  - Completely revised **21.12.3 "Wake-up Functions"**
- Additional minor corrections such as language and formatting updates have been incorporated throughout the document.