CAN Flexible Data-Rate (FD) Protocol Module

This section of the manual contains the following major topics:

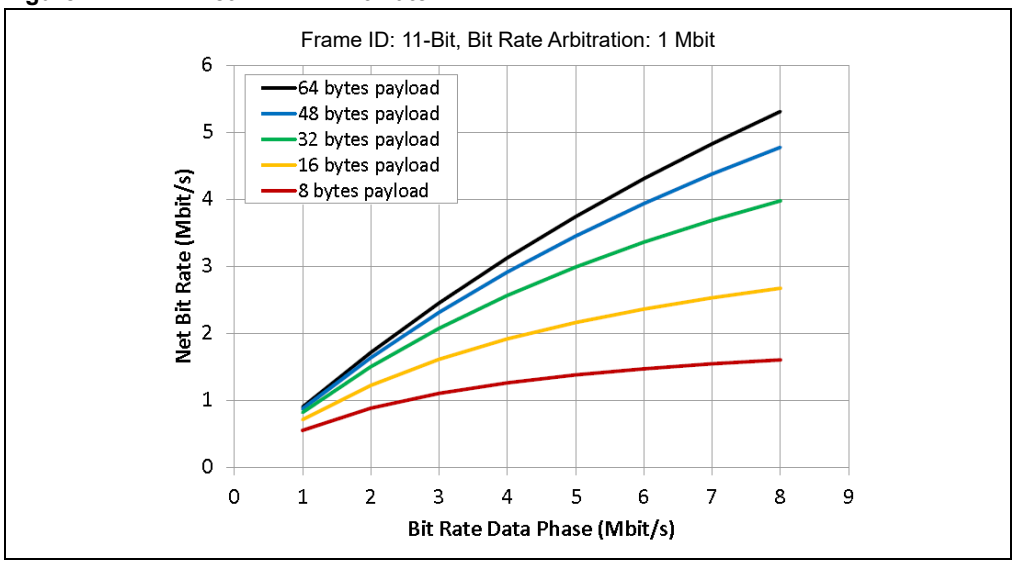
1.0 INTRODUCTION

CAN Flexible Data-Rate (FD) addresses the increasing demand for bandwidth on CAN buses. The two major enhancements over CAN 2.0B consist of:

- Increased data field of up to 64 data bytes (from a maximum eight data bytes for CAN 2.0B)
- Option to switch to faster bit rate after the arbitration field

Figure 1-1 shows the possible increase in net bit rate due to the higher Data Bit Rate (DBR) and increased data bytes per frame.

Figure 1-1: Net CAN FD Bit Rate



The CAN FD protocol is defined to allow CAN 2.0 messages and CAN FD messages to coexist on the same bus. This does not imply that non-CAN FD controllers can be mixed with CAN FD controllers on the same bus. Non-CAN FD controllers will generate error frames while receiving a CAN FD message.

1.1 Features

The CAN FD module has the following features:

General

- Nominal (Arbitration) Bit Rate up to 1 Mbps
- Data Bit Rate up to 8 Mbps
- CAN FD Controller modes:
  - Mixed CAN 2.0B and CAN FD mode
  - CAN 2.0B mode
- Conforms to ISO11898-1:2015

Message FIFOs

- 31 FIFOs Configurable as Transmit or Receive FIFOs
- One Transmit Queue (TXQ)
- Transmit Event FIFO (TEF) with 32-Bit Timestamp

Message Transmission

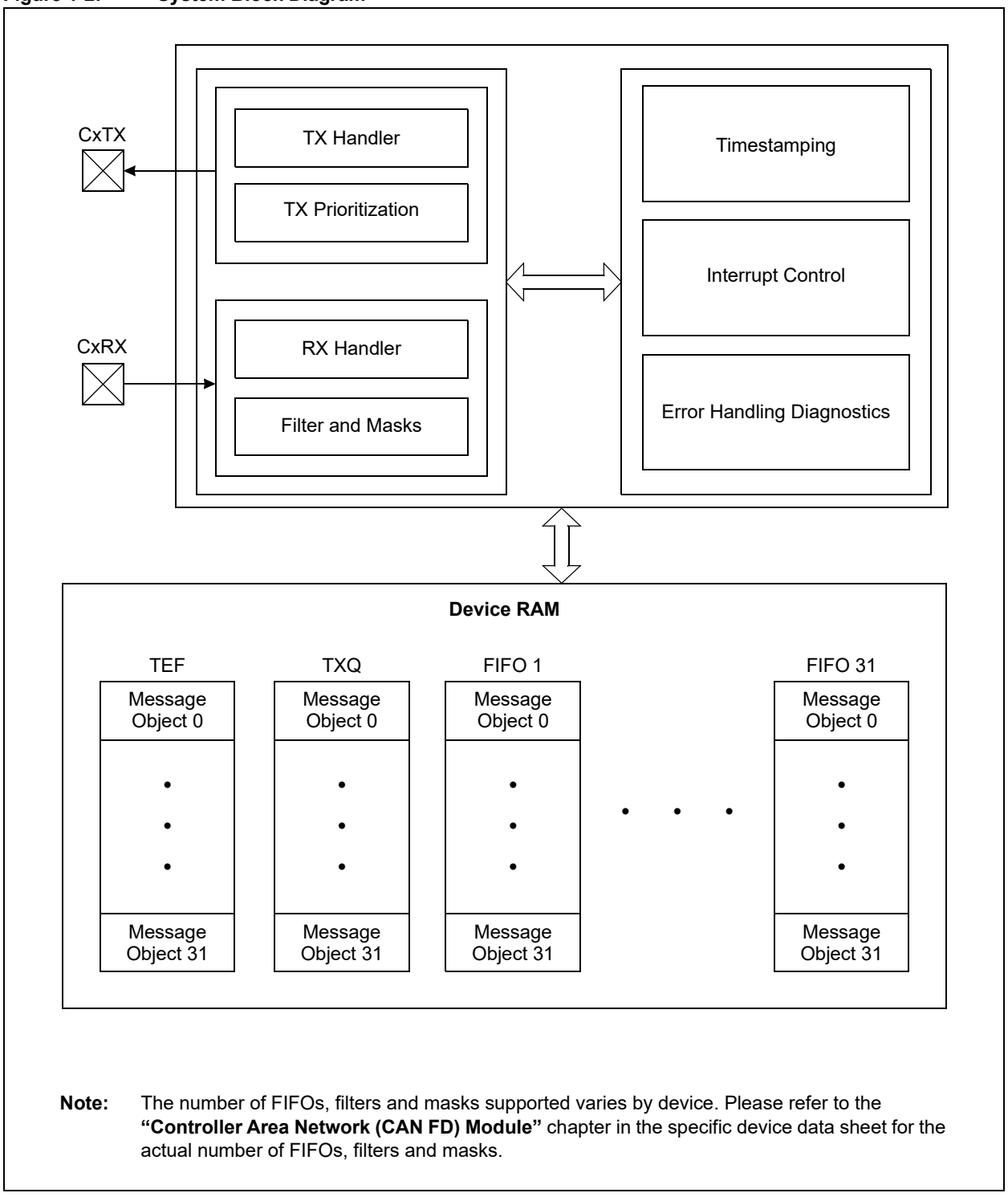
- Message Transmission Prioritization:
  - Based on priority bit field and/or
  - Message with lowest ID gets transmitted first using the TXQ
- Programmable Automatic Retransmission Attempts: Unlimited, Three Attempts or Disabled

Message Reception

- 32 Flexible Filter and Mask Objects
- Each Object can be Configured to Filter either:
  - Standard ID and first 18 data bits or
  - Extended ID
- 32-Bit Timestamp
- The CAN FD Bit Stream Processor (BSP) implements the Medium Access Control (MAC) of the CAN FD protocol described in ISO11898-1:2015. It serializes and deserializes the bit stream, encodes and decodes the CAN FD frames, manages the medium access, Acknowledges frames, and detects and signals errors.
- The TX handler prioritizes the messages that are requested for transmission by the transmit FIFOs. It uses the RAM interface to fetch the transmit data from RAM and provides it to the BSP for transmission.
- The BSP provides received messages to the RX handler. The RX handler uses an acceptance filter to filter the messages that shall be stored in the receive FIFOs. It uses the RAM interface to store received data into RAM.
- Each FIFO can be configured either as a transmit or receive FIFO. The FIFO control keeps track of the FIFO head and tail, and calculates the user address. In a TX FIFO, the user address points to the address in RAM where the data for the next transmit message is stored. In an RX FIFO, the user address points to the address in RAM where the data of the next receive message will be read. The user notifies the FIFO that a message is written to or read from RAM by incrementing the head/tail of the FIFO.
- The TXQ is a special transmit FIFO that transmits the messages based on the ID of the messages stored in the queue.
- The TEF stores the message IDs of the transmitted messages.
- A free-running Time Base Counter (TBC) is used to timestamp received messages. Messages in the TEF can also be timestamped.
- The CAN FD controller module generates interrupts when new messages are received or when messages are transmitted successfully.

Figure 1-2 shows the system block diagram.

Figure 1-2: System Block Diagram



Note: The number of FIFOs, filters and masks supported varies by device. Please refer to the **"Controller Area Network (CAN FD) Module"** chapter in the specific device data sheet for the actual number of FIFOs, filters and masks.

2.0 CAN FD MESSAGE FRAMES

The ISO11898-1:2015 describes the different CAN message frames in detail. Figure 2-1 through Figure 2-6 explain and summarize the construction of the messages and fields.

There are four different CAN data/remote frames (see Figure 2-2):

- **CAN Base Frame:** Classic CAN 2.0 frame using Standard ID
- **CAN FD Base Frame:** CAN FD frame using Standard ID
- **CAN Extended Frame:** Classic CAN 2.0 frame using Extended ID
- **CAN FD Extended Frame:** CAN FD frame using Extended ID

There are no remote frames in CAN FD frames; therefore, the RTR bit is replaced with the RRS bit (see Figure 2-2). The RRS bit in the CAN FD base frame can be used to extend the SID to 12 bits. When enabled, it is referred to as SID11; it is the Least Significant bit (LSb) of SID[11:0].

Figure 2-3 specifies the control field of the different CAN messages. Before CAN FD was added to the ISO11898-1:2015, the FDF bit was a reserved bit. Now the FDF bit selects between Classic and CAN FD formats.

The BRS bit selects if the bit rate should be switched in the data phase of CAN FD frames. Figure 2-6 illustrates the error and overload frames. These special frames do not change.

Note:	If an error is detected during the data phase of a CAN FD frame, the bit rate will be switched back to the Nominal Bit Rate (NBR). Error frames are always transmitted at the arbitration bit rate.

2.1 ISO vs. Non-ISO CRC

To support the system validation of non-ISO CRC ECUs, the CAN FD controller module supports both ISO CRC (according to ISO11898-1:2015) and non-ISO CRC (see Figure 2-4 and Figure 2-5). The CRC field is selectable using the ISOCRCEN bit (CxCONL[5]). The ISO CRC field contains the stuff count. This count was not included in the original CAN FD specification; it was added to fix a minor issue in the error detection of the original specification.

CAN FD frames use two different lengths of CRC: 17-bit for up to 16 data bytes and 21-bit for 20 or more data bytes. Technically, there are a total of six different CAN data/remote frames in the CAN FD.

Figure 2-1: General Data Frame

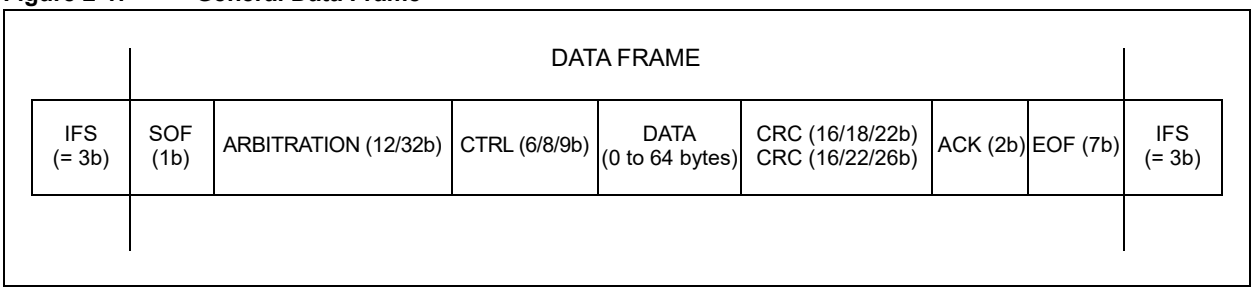


Figure 2-2: Arbitration Field

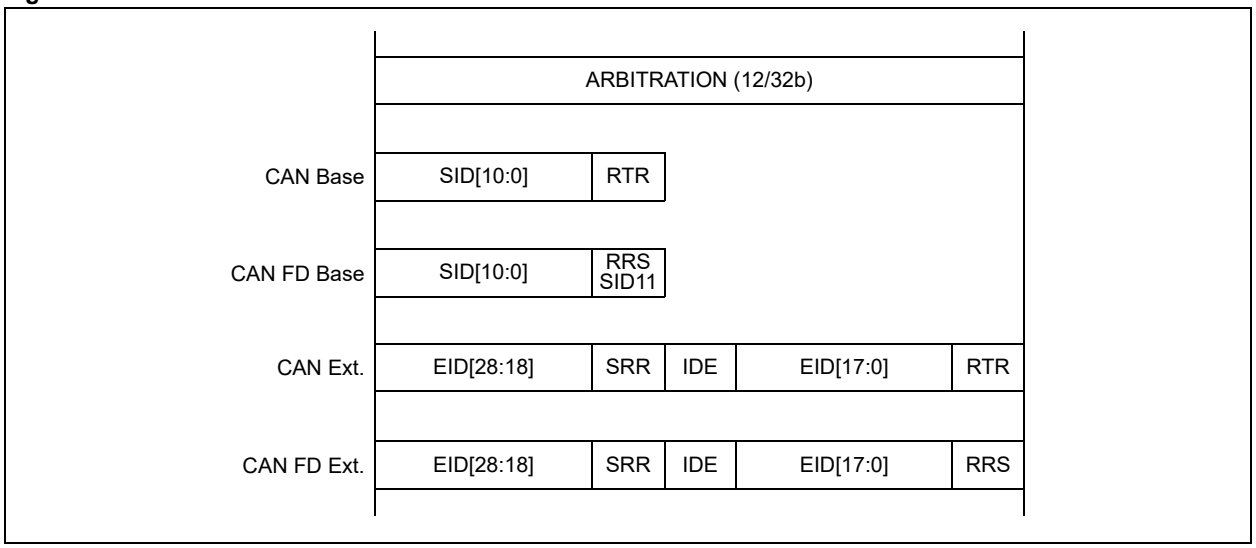


Figure 2-3: Control Field

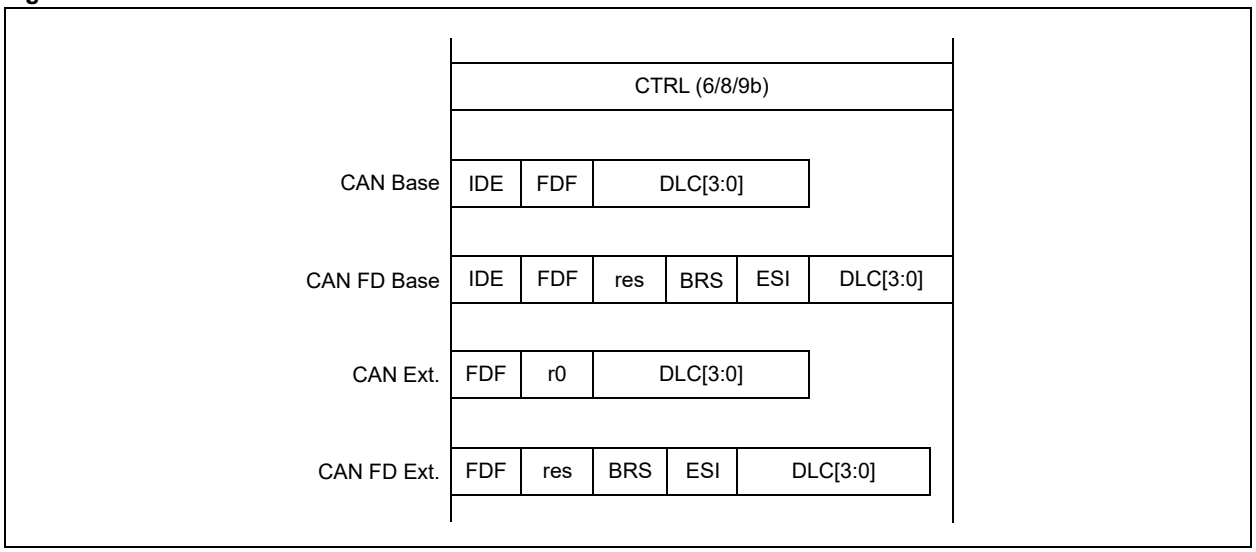


Figure 2-4: ISO CRC Field

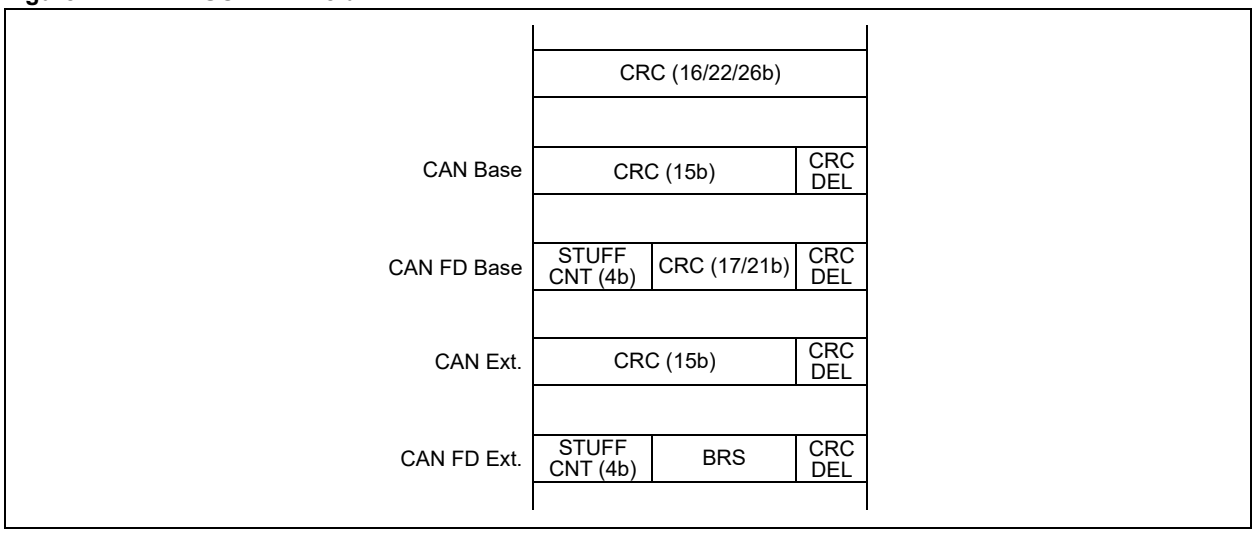


Figure 2-5: Non-ISO CRC Field

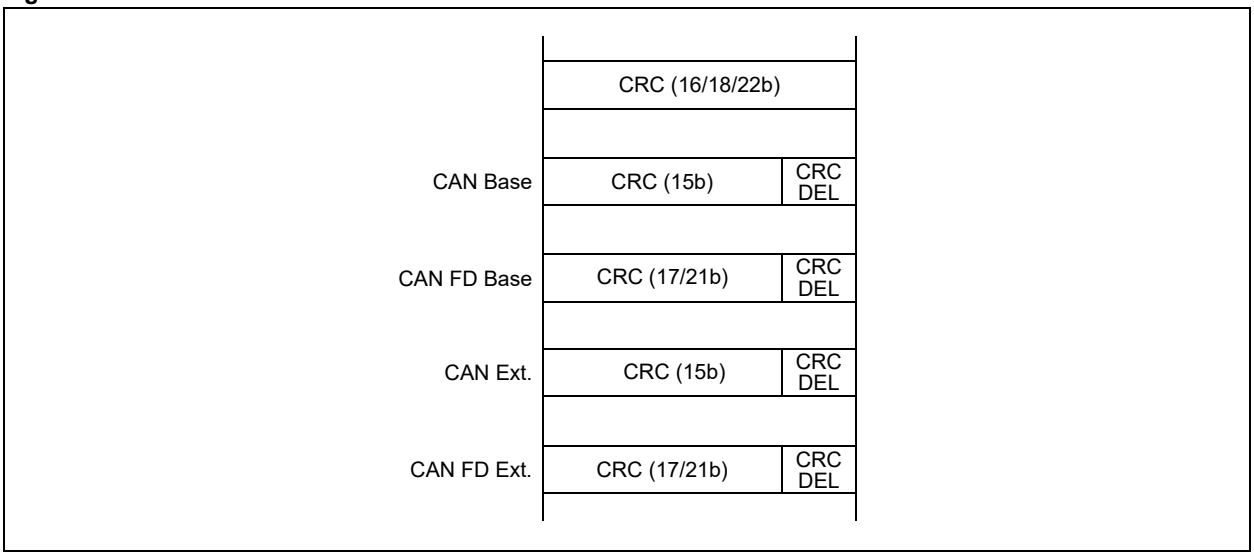
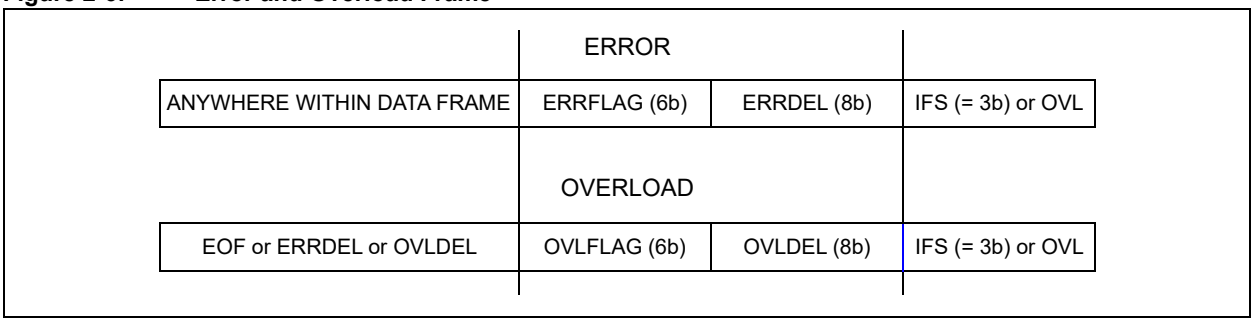


Figure 2-6: Error and Overload Frame



2.1.1 DLC ENCODING

The Data Length Code (DLC) specifies the number of data bytes a message frame contains. Table 2-1 illustrates the encoding.

Table 2-1: DLC Encoding

Frame	DLC	Number of Data Bytes
CAN 2.0 and CAN FD	0	0
	1	1
	2	2
	3	3
	4	4
	5	5
	6	6
	7	7
	8	8
CAN 2.0	9-15	8
CAN FD	9	12
	10	16
	11	20
	12	24
	13	32
	14	48
	15	64

3.0 CONTROL REGISTERS

CAN FD operations are controlled using the following Special Function Registers (SFRs). The following registers are described later in this section:

- Register 3-1: CxCONL
- Register 3-2: CxCONH
- Register 3-3: CxNBTCFGL
- Register 3-4: CxNBTCFGH
- Register 3-5: CxDBTCFGL
- Register 3-6: CxDBTCFGH
- Register 3-7: CxTDCL
- Register 3-8: CxTDCH
- Register 3-9: CxTBCL
- Register 3-10: CxTBCH
- Register 3-11: CxTSCONL
- Register 3-12: CxTSCONH
- Register 3-13: CxVECL
- Register 3-14: CxVECH
- Register 3-15: CxINTL
- Register 3-16: CxINTH
- Register 3-17: CxRXIFL
- Register 3-18: CxRXIFH
- Register 3-19: CxRXOVIFL
- Register 3-20: CxRXOVIFH
- Register 3-21: CxTXIFL
- Register 3-22: CxTXIFH
- Register 3-23: CxTXATIFL
- Register 3-24: CxTXATIFH
- Register 3-25: CxTXREQL
- Register 3-26: CxTXREQH
- Register 3-27: CxFIFOBAL
- Register 3-28: CxFIFOBAH
- Register 3-29: CxTXQCONL
- Register 3-30: CxTXQCONH
- Register 3-31: CxTXQSTA
- Register 3-32: CxFIFOCONxL
- Register 3-33: CxFIFOCONxH
- Register 3-34: CxFIFOSTAx
- Register 3-35: CxTEFCONL
- Register 3-36: CxTEFCONH
- Register 3-37: CxTEFSTA
- Register 3-38: CxFIFOUAxL
- Register 3-39: CxFIFOUAxH
- Register 3-40: CxTEFUAL
- Register 3-41: CxTEFUAH
- Register 3-42: CxTXQUAL
- Register 3-43: CxTXQUAH
- Register 3-44: CxTRECL
- Register 3-45: CxTRECH
- Register 3-46: CxBDIAG0L
- Register 3-47: CxBDIAG0H
- Register 3-48: CxBDIAG1L
- Register 3-49: CxBDIAG1H
- Register 3-50: CxFLTCONxL
- Register 3-51: CxFLTCONxH
- Register 3-52: CxFLTOBJxL
- Register 3-53: CxFLTOBJxH
- Register 3-54: CxMASKxL
- Register 3-55: CxMASKxH

Register 3-1: CxCONL: CAN Control Register Low

R/W-0	U-0	R/W-0	R/W-0	R-0	R/W-1	R/W-1	R/W-1
CON	—	SIDL	BRSDIS	BUSY	WFT1	WFT0	WAKFIL(1)
bit 15							bit 8

R/W-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
CLKSEL(1)	PXEDIS(1)	ISOCRCEN(1)	DNCNT4	DNCNT3	DNCNT2	DNCNT1	DNCNT0
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 **CON:** CAN Enable bit
1 = CAN module is enabled
0 = CAN module is disabled

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

bit 13 **SIDL:** CAN Stop in Idle Control bit
1 = Stops module operation in Idle mode
0 = Does not stop module operation in Idle mode

bit 12 **BRSDIS:** Bit Rate Switching (BRS) Disable bit
1 = Bit Rate Switching is disabled, regardless of BRS in the transmit message object
0 = Bit Rate Switching depends on BRS in the transmit message object

bit 11 **BUSY:** CAN Module is Busy bit
1 = The CAN module is active
0 = The CAN module is inactive

bit 10-9 **WFT[1:0]:** Selectable Wake-up Filter Time bits
11 = T11FILTER
10 = T10FILTER
01 = T01FILTER
00 = T00FILTER

bit 8 **WAKFIL:** Enable CAN Bus Line Wake-up Filter bit(1)
1 = Uses CAN bus line filter for wake-up
0 = CAN bus line filter is not used for wake-up

bit 7 **CLKSEL:** Module Clock Source Select bit(1)
1 = Auxiliary clock is active when module is enabled
0 = Clock from the CAN clock generator is active when the module is enabled

bit 6 **PXEDIS:** Protocol Exception Event Detection Disabled bit(1)
A recessive "reserved bit" following a recessive FDF bit is called a "Protocol Exception".
1 = Protocol exception is treated as a form error
0 = If a protocol exception is detected, CAN will enter the bus integrating state

bit 5 **ISOCRCEN:** Enable ISO CRC in CAN FD Frames bit(1)
1 = Includes Stuff Bit Count in CRC field and uses non-zero CRC initialization vector
0 = Does not include Stuff Bit Count in CRC field and uses CRC initialization vector with all zeros

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-1: CxCONL: CAN Control Register Low (Continued)

bit 4-0 **DNCNT[4:0]:** DeviceNet™ Filter Bit Number bits
10011-11111 = Invalid selection (compares up to 18 bits of data with EIDx)
10010 = Compares up to Data Byte 2, bit 6 with EID17
...
00001 = Compares up to Data Byte 0, bit 7 with EID0
00000 = Does not compare data bytes

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-2: CxCONH: CAN Control Register High

R/W-0	R/W-0	R/W-0	R/W-0	S/HC-0	R/W-1	R/W-0	R/W-0
TXBWS3	TXBWS2	TXBWS1	TXBWS0	ABAT	REQOP2	REQOP1	REQOP0
bit 15							bit 8

R-1	R-0	R-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0
OPMOD2	OPMOD1	OPMOD0	TXQEN(1)	STEF(1)	SERRLOM(1)	ESIGM(1)	RTXAT(1)
bit 7							bit 0

Legend:	S = Settable bit	HC = Hardware Clearable 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 **TXBWS[3:0]:** Transmit Bandwidth Sharing bits
1111-1100 = 4096
1011 = 2048
1010 = 1024
1001 = 512
1000 = 256
0111 = 128
0110 = 64
0101 = 32
0100 = 16
0011 = 8
0010 = 4
0001 = 2
0000 = No delay

bit 11 **ABAT:** Abort All Pending Transmissions bit
1 = Signals all transmit buffers to abort transmission
0 = Module will clear this bit when all transmissions are aborted

bit 10-8 **REQOP[2:0]:** Request Operation Mode bits
111 = Sets Restricted Operation mode
110 = Sets Normal CAN 2.0 mode; error frames on CAN FD frames
101 = Sets External Loopback mode
100 = Sets Configuration mode
011 = Sets Listen Only mode
010 = Sets Internal Loopback mode
001 = Sets Disable mode
000 = Sets Normal CAN FD mode; supports mixing of full CAN FD and Classic CAN 2.0 frames

bit 7-5 **OPMOD[2:0]:** Operation Mode Status bits
111 = Module is in Restricted Operation mode
110 = Module is in Normal CAN 2.0 mode; error frames on CAN FD frames
101 = Module is in External Loopback mode
100 = Module is in Configuration mode
011 = Module is in Listen Only mode
010 = Module is in Internal Loopback mode
001 = Module is in Disable mode
000 = Module is in Normal CAN FD mode; supports mixing of full CAN FD and Classic CAN 2.0 frames

bit 4 **TXQEN:** Enable Transmit Queue bit(1)
1 = Enables TXQ and reserves space in RAM
0 = Does not reserve space in RAM for TXQ

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-2: CxCONH: CAN Control Register High (Continued)

bit 3 **STEF:** Store in Transmit Event FIFO bit(1)
1 = Saves transmitted messages in TEF
0 = Does not save transmitted messages in TEF

bit 2 **SERRLOM:** Transition to Listen Only Mode on System Error bit(1)
1 = Transitions to Listen Only mode
0 = Transitions to Restricted Operation mode

bit 1 **ESIGM:** Transmit ESI in Gateway Mode bit(1)
1 = ESI is transmitted as recessive when the ESI of message is high or CAN controller is error passive
0 = ESI reflects error status of the CAN controller

bit 0 **RTXAT:** Restrict Retransmission Attempts bit(1)
1 = Restricted retransmission attempts, uses TXAT[1:0] (CxFIFOCONxH[6:5])
0 = Unlimited number of retransmission attempts, TXAT[1:0] bits will be ignored

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-3: CxNBTCFGL: CAN Nominal Bit Time Configuration Register Low(1)

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	TSEG2[6:0]						
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1
—	SJW[6:0]						
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 14-8 **TSEG2[6:0]:** Time Segment 2 bits (Phase Segment 2)
111 1111 = Length is 128 x TQ
...
000 0000 = Length is 1 x TQ

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

bit 6-0 **SJW[6:0]:** Synchronization Jump Width bits
111 1111 = Length is 128 x TQ
...
000 0000 = Length is 1 x TQ

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-4: CxNBTCFGH: CAN Nominal Bit Time Configuration Register High(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
BRP[7:0]							
bit 15							bit 8

R/W-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
TSEG1[7:0]							
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 **BRP[7:0]:** Baud Rate Prescaler bits
1111 1111 = TQ = 256/FCAN
...
0000 0000 = TQ = 1/FCAN

bit 7-0 **TSEG1[7:0]:** Time Segment 1 bits (Propagation Segment + Phase Segment 1)
1111 1111 = Length is 256 x TQ
...
0000 0000 = Length is 1 x TQ

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-5: CxDBTCFGL: CAN Data Bit Time Configuration Register Low(1)

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-1	R/W-1
—	—	—	—	TSEG2[3:0]			
bit 15							bit 8

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-1	R/W-1
—	—	—	—	SJW[3:0]			
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 15-12 **Unimplemented:** Read as '0'

bit 11-8 **TSEG2[3:0]:** Time Segment 2 bits (Phase Segment 2)
1111 = Length is 16 x TQ
...
0000 = Length is 1 x TQ

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

bit 3-0 **SJW[3:0]:** Synchronization Jump Width bits
1111 = Length is 16 x TQ
...
0000 = Length is 1 x TQ

Note 1: This register can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-6: CxDBTCFGH: CAN Data Bit Time Configuration Register High(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
BRP[7:0]							
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-1	R/W-1	R/W-1	R/W-0
—	—	—	TSEG1[4:0]				
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 **BRP[7:0]:** Baud Rate Prescaler bits
1111 1111 = TQ = 256/FCAN
...
0000 0000 = TQ = 1/FCAN

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

bit 4-0 **TSEG1[4:0]:** Time Segment 1 bits (Propagation Segment + Phase Segment 1)
1 1111 = Length is 32 x TQ
...
0 0000 = Length is 1 x TQ

Note 1: This register can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-7: CxTDCL: CAN Transmitter Delay Compensation Register Low(1)

U-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0
—	TDCO[6:0]						
bit 15							bit 8

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	TDCV[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 **Unimplemented:** Read as '0'

bit 14-8 **TDCO[6:0]:** Transmitter Delay Compensation Offset bits (Secondary Sample Point (SSP))
111 1111 = -64 x TCAN
...
011 1111 = 63 x TCAN
...
000 0000 = 0 x TCAN

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

bit 5-0 **TDCV[5:0]:** Transmitter Delay Compensation Value bits (Secondary Sample Point (SSP))
11 1111 = 63 x TCAN
...
00 0000 = 0 x TCAN

Note 1: This register can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-8: CxTDCH: CAN Transmitter Delay Compensation Register High(1)

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
—	—	—	—	—	—	EDGFLTEN	SID11EN
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	U-0	R/W-1	R/W-0
—	—	—	—	—	—	TDCMOD1	TDCMOD0
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 **Unimplemented:** Read as '0'

bit 9 **EDGFLTEN:** Enable Edge Filtering During Bus Integration State bit
1 = Edge filtering is enabled according to ISO11898-1:2015
0 = Edge filtering is disabled

bit 8 **SID11EN:** Enable 12-Bit SID in CAN FD Base Format Messages bit
1 = RRS is used as SID11 in CAN FD base format messages: SID[11:0] = {SID[10:0], SID11}
0 = Does not use RRS; SID[10:0]

bit 7-2 **Unimplemented:** Read as '0'

bit 1-0 **TDCMOD[1:0]:** Transmitter Delay Compensation mode bits (Secondary Sample Point (SSP))
10-11 = Auto: Measures delay and adds TDCO
01 = Manual: Does not measure, uses TDCV[5:0] + TDCO[6:0] from register
00 = Disables

Note 1: This register can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-9: CxTBCL: CAN Time Base Counter Register Low(1,2)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TBC[15:8]							
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
TBC[7:0]							
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 15-0 **TBC[15:0]** CAN Time Base Counter bits
This is a free-running timer that increments every TBCPRE[9:0] clock when TBCEN is set.

Note 1: The TBC will be stopped and reset when TBCEN = 0 to save power.
2: The TBC prescaler count will be reset on any write to CxTBCL/H (TBCPREx will be unaffected).

Register 3-10: CxTBCH: CAN Time Base Counter Register High(1,2)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TBC[31:24]							
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
TBC[23:16]							
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 15-0 **TBC[31:16]** CAN Time Base Counter bits
This is a free-running timer that increments every TBCPRE[9:0] clock when TBCEN is set.

Note 1: The Time Base Counter (TBC) will be stopped and reset when TBCEN = 0 to save power.
2: The TBC prescaler count will be reset on any write to CxTBCL/H (TBCPREx will be unaffected).

Register 3-11: CxTSCONL: CAN Timestamp Control Register Low

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
—	—	—	—	—	—	TBCPRE[9:8]	
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
TBCPRE[7:0]							
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 15-10 **Unimplemented:** Read as '0'

bit 9-0 **TBCPRE[9:0]:** CAN Time Base Counter Prescaler bits
1023 = TBC increments every 1024 clocks
...
0 = TBC increments every 1 clock

Register 3-12: CxTSCONH: CAN Timestamp Control Register High

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	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	TSRES	TSEOF	TBCEN
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 15-3 **Unimplemented:** Read as '0'

bit 2 **TSRES:** Timestamp Reset bit (CAN FD frames only)
1 = At sample point of the bit following the FDF bit
0 = At sample point of Start-of-Frame (SOF)

bit 1 **TSEOF:** Timesstamp End-of-Frame (EOF) bit
1 = Timestamp when frame is taken valid (11898-1 10.7):
- RX no error until last, but one bit of EOF
- TX no error until the end of EOF

0 = Timestamp at "beginning" of frame:
- Classical Frame: At sample point of SOF
- FD Frame: see TSRES bit

bit 0 **TBCEN:** Time Base Counter (TBC) Enable bit
1 = Enables TBC
0 = Stops and resets TBC

Register 3-13: CxVECL: CAN Interrupt Code Register Low

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:			
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
11111 = Filter 31
11110 = Filter 30
...
00001 = Filter 1
00000 = Filter 0

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

bit 6-0 **ICODE[6:0]:** Interrupt Flag Code bits
1001011-1111111 = Reserved
1001010 = Transmit attempt interrupt (any bit in CxTXATIF is set)
1001001 = Transmit event FIFO interrupt (any bit in CxTEFSTA is set)
1001000 = Invalid message occurred (IVMIF/IE)
1000111 = CAN module mode change occurred (MODIF/IE)
1000110 = CAN timer overflow (TBCIF/IE)
1000101 = RX/TX MAB overflow/underflow (RX: Message received before previous message was saved to memory; TX: Can't feed TX MAB fast enough to transmit consistent data) (SERRIF/IE)
1000100 = Address error interrupt (illegal FIFO address presented to system) (SERRIF/IE)
1000011 = Receive FIFO overflow interrupt (any bit in CxRXOVIF is set)
1000010 = Wake-up interrupt (WAKIF/WAKIE)
1000001 = Error interrupt (CERRIF/IE)
1000000 = No interrupt
0100000-0111111 = Reserved
0011111 = FIFO 31 interrupt (TFIF31 or RFIF31 is set)
...
0000001 = FIFO 1 Interrupt (TFIF1 or RFIF1 is set)
0000000 = FIFO 0 Interrupt (TFIF0 is set)

Register 3-14: CxVECH: CAN Interrupt Code Register High

U-0	R-1	R-0	R-0	R-0	R-0	R-0	R-0
—	RXCODE[6:0]						
bit 15							bit 8

U-0	R-1	R-0	R-0	R-0	R-0	R-0	R-0
—	TXCODE[6:0]						
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 14-8 **RXCODE[6:0]:** Receive Interrupt Flag Code bits
1000001-1111111 = Reserved
1000000 = No interrupt
0100000-0111111 = Reserved
0011111 = FIFO 31 interrupt (RFIF[31] is set)
...
0000010 = FIFO 2 interrupt (RFIF[2] is set)
0000001 = FIFO 1 interrupt (RFIF[1] is set)
0000000 = Reserved; FIFO 0 cannot receive

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

bit 6-0 **TXCODE[6:0]:** Transmit Interrupt Flag Code bits
1000001-1111111 = Reserved
1000000 = No interrupt
0100000-0111111 = Reserved
0011111 = FIFO 31 interrupt (TFIF[31] is set)
...
0000001 = FIFO 1 interrupt (TFIF[1] is set)
0000000 = FIFO 0 interrupt (TFIF[0] is set)

Register 3-15: CxINTL: CAN Interrupt Register Low

HS/C-0	HS/C-0	HS/C-0	HS/C-0	R-0	R-0	U-0	U-0
IVMIF(1)	WAKIF(1)	CERRIF(1)	SERRIF(1)	RXOVIF	TXATIF	—	—
bit 15							bit 8

U-0	U-0	U-0	R-0	HS/C-0	HS/C-0	R-0	R-0
—	—	—	TEFIF	MODIF(1)	TBCIF(1)	RXIF	TXIF
bit 7							bit 0

Legend:	HS = Hardware Settable bit	C = Clearable 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 **IVMIF:** Invalid Message Interrupt Flag bit(1)
1 = Invalid message interrupt occurred
0 = No invalid message interrupt

bit 14 **WAKIF:** Bus Wake-up Activity Interrupt Flag bit(1)
1 = Wake-up activity interrupt occurred
0 = No wake-up activity interrupt

bit 13 **CERRIF:** CAN Bus Error Interrupt Flag bit(1)
1 = CAN bus error interrupt occurred
0 = No CAN bus error interrupt

bit 12 **SERRIF:** System Error Interrupt Flag bit(1)
1 = System error interrupt occurred
0 = No system error interrupt

bit 11 **RXOVIF:** Receive Buffer Overflow Interrupt Flag bit
1 = Receive buffer overflow interrupt occurred
0 = No receive buffer overflow interrupt

bit 10 **TXATIF:** Transmit Attempt Interrupt Flag bit
1 = Transmit attempt interrupt occurred
0 = No transmit attempt interrupt

bit 9-5 **Unimplemented:** Read as '0'

bit 4 **TEFIF:** Transmit Event FIFO Interrupt Flag bit
1 = Transmit event FIFO interrupt occurred
0 = No transmit event FIFO

bit 3 **MODIF:** CAN Mode Change Interrupt Flag bit(1)
1 = CAN module mode change occurred (OPMOD[2:0] have changed to reflect REQOP[2:0])
0 = No mode change occurred

bit 2 **TBCIF:** CAN Timer Overflow Interrupt Flag bit(1)
1 = TBC has overflowed
0 = TBC has not overflow

bit 1 **RXIF:** Receive Object Interrupt Flag bit
1 = Receive object interrupt is pending
0 = No receive object interrupts are pending

bit 0 **TXIF:** Transmit Object Interrupt Flag bit
1 = Transmit object interrupt is pending
0 = No transmit object interrupts are pending

Note 1: CxINTL: Flags are set by hardware and cleared by application.

Register 3-16: CxINTH: CAN Interrupt Register High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0
IVMIE	WAKIE	CERRIE	SERRIE	RXOVIE	TXATIE	—	—
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
—	—	—	TEFIE	MODIE	TBCIE	RXIE	TXIE
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 **IVMIE:** Invalid Message Interrupt Enable bit
1 = Invalid message interrupt is enabled
0 = Invalid message interrupt is disabled

bit 14 **WAKIE:** Bus Wake-up Activity Interrupt Enable bit
1 = Wake-up activity interrupt is enabled
0 = Wake-up activity interrupt is disabled

bit 13 **CERRIE:** CAN Bus Error Interrupt Enable bit
1 = CAN bus error interrupt is enabled
0 = CAN bus error interrupt is disabled

bit 12 **SERRIE:** System Error Interrupt Enable bit
1 = System error interrupt is enabled
0 = System error interrupt is disabled

bit 11 **RXOVIE:** Receive Buffer Overflow Interrupt Enable bit
1 = Receive buffer overflow interrupt is enabled
0 = Receive buffer overflow interrupt is disabled

bit 10 **TXATIE:** Transmit Attempt Interrupt Enable bit
1 = Transmit attempt interrupt is enabled
0 = Transmit attempt interrupt is disabled

bit 9-5 **Unimplemented:** Read as '0'

bit 4 **TEFIE:** Transmit Event FIFO Interrupt Enable bit
1 = Transmit event FIFO interrupt is enabled
0 = Transmit event FIFO interrupt is disabled

bit 3 **MODIE:** Mode Change Interrupt Enable bit
1 = Mode change interrupt is enabled
0 = Mode change interrupt is disabled

bit 2 **TBCIE:** CAN Timer Interrupt Enable bit
1 = CAN timer interrupt is enabled
0 = CAN timer interrupt is disabled

bit 1 **RXIE:** Receive Object Interrupt Enable bit
1 = Receive object interrupt is enabled
0 = Receive object interrupt is disabled

bit 0 **TXIE:** Transmit Object Interrupt Enable bit
1 = Transmit object interrupt is enabled
0 = Transmit object interrupt is disabled

Register 3-17: CxRXIFL: CAN Receive Interrupt Status Register Low(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFIF[15:8]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	U-0
RFIF[7:1]							—
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 15-1 **RFIF[15:1]:** Receive FIFO Interrupt Pending bits
1 = One or more enabled receive FIFO interrupts are pending
0 = No enabled receive FIFO interrupts are pending

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

Note 1: CxRXIFL: FIFO: RFIFx = 'or' of enabled RX FIFO flags (flags need to be cleared in the FIFO register).

Register 3-18: CxRXIFH: CAN Receive Interrupt Status Register High(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFIF[31:24]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFIF[23:16]							
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 15-0 **RFIF[31:16]:** Receive FIFO Interrupt Pending bits
1 = One or more enabled receive FIFO interrupts are pending
0 = No enabled receive FIFO interrupts are pending

Note 1: CxRXIFH: FIFO: RFIFx = 'or' of enabled RX FIFO flags (flags need to be cleared in the FIFO register).

Register 3-19: CxRXOVIFL: CAN Receive Overflow Interrupt Status Register Low(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFOVIF[15:8]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	U-0
RFOVIF[7:1]							—
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 15-1 **RFOVIF[15:1]:** Receive FIFO Overflow Interrupt Pending bits
1 = Interrupt is pending
0 = Interrupt is not pending

bit 0 **Unimplemented:** Read as ‘0’

Note 1: CxRXOVIFL: FIFO: RFOVIFx (flag needs to be cleared in the FIFO register).

Register 3-20: CxRXOVIFH: CAN Receive Overflow Interrupt Status Register High(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFOVIF[31:24]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
RFOVIF[23:16]							
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 15-0 **RFOVIF[31:16]:** Receive FIFO Overflow Interrupt Pending bits
1 = Interrupt is pending
0 = Interrupt is not pending

Note 1: CxRXOVIFH: FIFO: RFOVIFx (flag needs to be cleared in the FIFO register).

Register 3-21: CxTXIFL: CAN Transmit Interrupt Status Register Low(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFIF[15:8]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFIF[7:0](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 15-0 **TFIF[15:0]:** Transmit FIFO/TXQ Interrupt Pending bits(2)
1 = One or more enabled transmit FIFO/TXQ interrupts are pending
0 = No enabled transmit FIFO/TXQ interrupts are pending

Note 1: CxTXIFL: FIFO: TFIFx = 'or' of the enabled TX FIFO flags (flags need to be cleared in the FIFO register).
2: TFIF0 is for the TXQ.

Register 3-22: CxTXIFH: CAN Transmit Interrupt Status Register High(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFIF[31:24]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFIF[23:16]							
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 15-0 **TFIF[31:16]:** Transmit FIFO/TXQ Interrupt Pending bits
1 = One or more enabled transmit FIFO/TXQ interrupts are pending
0 = No enabled transmit FIFO/TXQ interrupts are pending

Note 1: CxTXIFH: FIFO: TFIFx = 'or' of the enabled TX FIFO flags (flags need to be cleared in the FIFO register).

Register 3-23: CxTXATIFL: CAN Transmit Attempt Interrupt Status Register Low(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFATIF[15:8]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFATIF[7:0](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 15-0 **TFATIF[15:0]:** Transmit FIFO/TXQ Attempt Interrupt Pending bits(2)
1 = Interrupt is pending
0 = Interrupt is not pending

Note 1: CxTXATIFL: FIFO: TFATIFx (flag needs to be cleared in the FIFO register).
2: TFATIF0 is for the TXQ.

Register 3-24: CxTXATIFH: CAN Transmit Attempt Interrupt Status Register High(1)

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFATIF[31:24]							
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
TFATIF[23:16]							
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 15-0 **TFATIF[31:16]:** Transmit FIFO/TXQ Attempt Interrupt Pending bits
1 = Interrupt is pending
0 = Interrupt is not pending

Note 1: CxTXATIFH: FIFO: TFATIFx (flag needs to be cleared in the FIFO register).

Register 3-25: CxTXREQL: CAN Transmit Request Register Low

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXREQ[15:8]							
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
TXREQ[7:1]							TXREQ0
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 15-1 **TXREQ[15:0]:** Message Send Request bits

TXEN = 1 (object configured as a transmit object):
Setting this bit to '1' requests sending a message. The bit will automatically clear when the message(s) queued in the object is (are) successfully sent. This bit can NOT be used for aborting a transmission.

TXEN = 0 (object configured as a receive object):
This bit has no effect.

bit 0 **TXREQ0:** Transmit Queue Message Send Request bit

Setting this bit to '1' requests sending a message. The bit will automatically clear when the message(s) queued in the object is (are) successfully sent. This bit can NOT be used for aborting a transmission.

Register 3-26: CxTXREQH: CAN Transmit Request Register High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXREQ[31:24]							
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
TXREQ[23:16]							
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 15-0 **TXREQ[31:16]:** Message Send Request bits

TXEN = 1 (object configured as a transmit object):
Setting this bit to '1' requests sending a message. The bit will automatically clear when the message(s) queued in the object is (are) successfully sent. This bit can NOT be used for aborting a transmission.

TXEN = 0 (object configured as a receive object):
This bit has no effect.

Register 3-27: CxFIFOBAL: CAN Message Memory Base Address Register Low

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FIFOBA[15:8]							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0(1)	U-0(1)
FIFOBA[7:0]							
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 15-0 **FIFOBA[15:0]:** Message Memory Base Address bits
Defines the base address for the transmit event FIFO followed by the message objects.

Note 1: Bits[1:0] are forced to '0' to be 32-bit word aligned.

Register 3-28: CxFIFOBAH: CAN Message Memory Base Address Register High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FIFOBA[31:24]							
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
FIFOBA[23:16]							
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 15-0 **FIFOBA[31:16]:** Message Memory Base Address bits
Defines the base address for the transmit event FIFO followed by the message objects.

Register 3-29: CxTXQCONL: CAN Transmit Queue Control Register Low

U-0	U-0	U-0	U-0	U-0	S/HC-1	R/W/HC-0	S/HC-0
—	—	—	—	—	FRESET	TXREQ	UINC
bit 15							bit 8

R-1	U-0	U-0	R/W-0	U-0	R/W-0	U-0	R/W-0
TXEN(1)	—	—	TXATIE	—	TXQEIE	—	TXQNIE
bit 7							bit 0

Legend:	S = Settable bit	HC = Hardware Clearable 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 15-11 **Unimplemented:** Read as '0'

bit 10 **FRESET:** FIFO Reset bit
- 1 = FIFO will be reset when bit is set, cleared by hardware when FIFO is reset; user should poll whether this bit is clear before taking any action
- 0 = No effect

bit 9 **TXREQ:** Message Send Request bit
- 1 = Requests sending a message; the bit will automatically clear when all the messages queued in the TXQ are successfully sent
- 0 = Clearing the bit to '0' while set ('1') will request a message abort

bit 8 **UINC:** Increment Head/Tail bit

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

bit 7 **TXEN:** TX Enable bit(1)

1 = Transmit Message Queue. This bit always reads as '1'.

bit 6-5 **Unimplemented:** Read as '0'

bit 4 **TXATIE:** Transmit Attempts Exhausted Interrupt Enable bit
- 1 = Enables interrupt
- 0 = Disables interrupt

bit 3 **Unimplemented:** Read as '0'

bit 2 **TXQEIE:** Transmit Queue Empty Interrupt Enable bit
- 1 = Interrupt is enabled for TXQ empty
- 0 = Interrupt is disabled for TXQ empty

bit 1 **Unimplemented:** Read as '0'

bit 0 **TXQNIE:** Transmit Queue Not Full Interrupt Enable bit
- 1 = Interrupt is enabled for TXQ not full
- 0 = Interrupt is disabled for TXQ not full

Note 1: Please refer to the specific device data sheet for the Reset value of the TXEN bit.

Register 3-30: CxTXQCONH: CAN Transmit Queue Control Register High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PLSIZE2(1)	PLSIZE1(1)	PLSIZE0(1)	FSIZE4(1)	FSIZE3(1)	FSIZE2(1)	FSIZE1(1)	FSIZE0(1)
bit 15							bit 8

U-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TXAT1	TXAT0	TXPRI4	TXPRI3	TXPRI2	TXPRI1	TXPRI0
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 **PLSIZE[2:0]:** Payload Size bits(1)
111 = 64 data bytes
110 = 48 data bytes
101 = 32 data bytes
100 = 24 data bytes
011 = 20 data bytes
010 = 16 data bytes
001 = 12 data bytes
000 = 8 data bytes

bit 12-8 **FSIZE[4:0]:** FIFO Size bits(1)
11111 = FIFO is 32 messages deep
...
00010 = FIFO is 3 messages deep
00001 = FIFO is 2 messages deep
00000 = FIFO is 1 message deep

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

bit 6-5 **TXAT[1:0]:** Retransmission Attempts bits
This feature is enabled when RTXAT (CxCONH[0]) is set.
11 = Unlimited number of retransmission attempts
10 = Unlimited number of retransmission attempts
01 = Three retransmission attempts
00 = Disable retransmission attempts

bit 4-0 **TXPRI[4:0]:** Message Transmit Priority bits
11111 = Highest message priority
...
00000 = Lowest message priority

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-31: CxTXQSTA: CAN Transmit Queue Status Register

U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
—	—	—	TXQCI4(1)	TXQCI3(1)	TXQCI2(1)	TXQCI1(1)	TXQCI0(1)
bit 15							bit 8

R-0	R-0	R-0	HS/C-0	U-0	R-1	U-0	R-1
TXABT(2)	TXLARB(2)	TXERR(2)	TXATIF	—	TXQEIF	—	TXQNIF
bit 7							bit 0

Legend:	C = Clearable bit	HS = Hardware Settable 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 **TXQCI[4:0]:** Transmit Queue Message Index bits(1)
A read of this register will return an index to the message that the FIFO will next attempt to transmit.

bit 7 **TXABT:** Message Aborted Status bit(2)
1 = Message was aborted
0 = Message completed successfully

bit 6 **TXLARB:** Message Lost Arbitration Status bit(2)
1 = Message lost arbitration while being sent
0 = Message did not lose arbitration while being sent

bit 5 **TXERR:** Error Detected During Transmission bit(2)
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 4 **TXATIF:** Transmit Attempts Exhausted Interrupt Pending bit
1 = Interrupt is pending
0 = Interrupt is not pending

bit 3 **Unimplemented:** Read as '0'

bit 2 **TXQEIF:** Transmit Queue Empty Interrupt Flag bit
1 = TXQ is empty
0 = TXQ is not empty, at least 1 message is queued to be transmitted

bit 1 **Unimplemented:** Read as '0'

bit 0 **TXQNIF:** Transmit Queue Not Full Interrupt Flag bit
1 = TXQ is not full
0 = TXQ is full

Note 1: The TXQCI[4:0] bits give a zero-indexed value to the message in the TXQ. If the TXQ is four messages deep (FSIZE = 3), TXQCIx will take on a value of 0 to 3, depending on the state of the TXQ.

2: These bits are updated when a message completes (or aborts) or when the TXQ is reset.

Register 3-32: CxFIFOCONxL: CAN FIFO Control Register x (x = 1 to 31) Low

U-0	U-0	U-0	U-0	U-0	S/HC-1	R/W/HC-0	S/HC-0
—	—	—	—	—	FRESET	TXREQ	UINC
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
TXEN	RTREN	RXTSEN(1)	TXATIE	RXOVIE	TFERFFIE	TFHRFHIE	TFNRFNIE
bit 7							bit 0

Legend:	S = Settable bit	HC = Hardware Clearable 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 15-11 **Unimplemented:** Read as '0'

bit 10 **FRESET:** FIFO Reset bit
1 = FIFO will be reset when bit is set, cleared by hardware when FIFO is reset; user should poll whether this bit is clear before taking any action
0 = No effect

bit 9 **TXREQ:** Message Send Request bit
TXEN = 1 (FIFO configured as a transmit FIFO):
1 = Requests sending a message; the bit will automatically clear when all the messages queued in the FIFO are successfully sent
0 = Clearing the bit to '0' while set ('1') will request a message abort
TXEN = 0 (FIFO configured as a receive FIFO):
This bit has no effect.

bit 8 **UINC:** Increment Head/Tail bit
TXEN = 1 (FIFO configured as a transmit FIFO):
When this bit is set, the FIFO head will increment by a single message.
TXEN = 0 (FIFO configured as a receive FIFO):
When this bit is set, the FIFO tail will increment by a single message.

bit 7 **TXEN:** TX/RX Buffer Selection bit
1 = Transmits message object
0 = Receives message object

bit 6 **RTREN:** Auto-Remote Transmit (RTR) Enable bit
1 = When a Remote Transmit is received, TXREQ will be set
0 = When a Remote Transmit is received, TXREQ will be unaffected

bit 5 **RXTSEN:** Received Message Timestamp Enable bit(1)
1 = Captures timestamp in received message object in RAM
0 = Does not capture time stamp

bit 4 **TXATIE:** Transmit Attempts Exhausted Interrupt Enable bit
1 = Enables interrupt
0 = Disables interrupt

bit 3 **RXOVIE:** Overflow Interrupt Enable bit
1 = Interrupt is enabled for overflow event
0 = Interrupt is disabled for overflow event

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-32: CxFIFOCONxL: CAN FIFO Control Register x (x = 1 to 31) Low (Continued)

bit 2 **TFERFFIE:** Transmit/Receive FIFO Empty/Full Interrupt Enable bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Empty Interrupt Enable
1 = Interrupt is enabled for FIFO empty
0 = Interrupt is disabled for FIFO empty
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Full Interrupt Enable
1 = Interrupt is enabled for FIFO full
0 = Interrupt is disabled for FIFO full

bit 1 **TFHRFHIE:** Transmit/Receive FIFO Half Empty/Half Full Interrupt Enable bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Half Empty Interrupt Enable
1 = Interrupt is enabled for FIFO half empty
0 = Interrupt is disabled for FIFO half empty
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Half Full Interrupt Enable
1 = Interrupt is enabled for FIFO half full
0 = Interrupt is disabled for FIFO half full

bit 0 **TFNRFNIE:** Transmit/Receive FIFO Not Full/Not Empty Interrupt Enable bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Not Full Interrupt Enable
1 = Interrupt is enabled for FIFO not full
0 = Interrupt is disabled for FIFO not full
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Not Empty Interrupt Enable
1 = Interrupt is enabled for FIFO not empty
0 = Interrupt is disabled for FIFO not empty

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-33: CxFIFOCONxH: CAN FIFO Control Register x (x = 1 to 31) High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PLSIZE2(1)	PLSIZE1(1)	PLSIZE0(1)	FSIZE4(1)	FSIZE3(1)	FSIZE2(1)	FSIZE1(1)	FSIZE0(1)
bit 15							bit 8

U-0	R/W-1	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TXAT1	TXAT0	TXPRI4	TXPRI3	TXPRI2	TXPRI1	TXPRI0
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 **PLSIZE[2:0]:** Payload Size bits(1)
111 = 64 data bytes
110 = 48 data bytes
101 = 32 data bytes
100 = 24 data bytes
011 = 20 data bytes
010 = 16 data bytes
001 = 12 data bytes
000 = 8 data bytes

bit 12-8 **FSIZE[4:0]:** FIFO Size bits(1)
11111 = FIFO is 32 messages deep
...
00010 = FIFO is 3 messages deep
00001 = FIFO is 2 messages deep
00000 = FIFO is 1 message deep

bit 7 **Unimplemented:** Read as ‘0’

bit 6-5 **TXAT[1:0]:** Retransmission Attempts bits
This feature is enabled when RTXAT (CxCONH[0]) is set.
11 = Unlimited number of retransmission attempts
10 = Unlimited number of retransmission attempts
01 = Three retransmission attempts
00 = Disables retransmission attempts

bit 4-0 **TXPRI[4:0]:** Message Transmit Priority bits
11111 = Highest message priority
...
00000 = Lowest message priority

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-34: CxFIFOSTAx: CAN FIFO Status Register x (x = 1 to 31)

U-0	U-0	U-0	R-0	R-0	R-0	R-0	R-0
—	—	—	FIFOCI4[1]	FIFOCI3[1]	FIFOCI2[1]	FIFOCI1[1]	FIFOCI0[1]
bit 15							bit 8

R-0	R-0	R-0	HS/C-0	HS/C-0	R-0	R-0	R-0
TXABT[3]	TXLARB[2]	TXERR[2]	TXATIF	RXOVIF	TFERFFIF	TFHRFHIF	TFNRFNIF
bit 7							bit 0

Legend:	C = Clearable bit	HS = Hardware Settable 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 **FIFOCI[4:0]:** FIFO Message Index bits[1]

TXEN = 1 (FIFO configured as a transmit buffer):
A read of this register will return an index to the message that the FIFO will next attempt to transmit.

TXEN = 0 (FIFO configured as a receive buffer):
A read of this register will return an index to the message that the FIFO will use to save the next message.

bit 7 **TXABT:** Message Aborted Status bit[3]

1 = Message was aborted
0 = Message completed successfully

bit 6 **TXLARB:** Message Lost Arbitration Status bit[2]

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

bit 5 **TXERR:** Error Detected During Transmission bit[2]

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 4 **TXATIF:** Transmit Attempts Exhausted Interrupt Pending bit

TXEN = 1 (FIFO configured as a transmit buffer):
1 = Interrupt is pending
0 = Interrupt is not pending

TXEN = 0 (FIFO configured as a receive buffer):
Unused, reads as '0'.

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

TXEN = 1 (FIFO configured as a transmit buffer):
Unused, reads as '0'.

TXEN = 0 (FIFO configured as a receive buffer):
1 = Overflow event has occurred
0 = No overflow event occurred

Note 1: FIFOCI[4:0] gives a zero-indexed value to the message in the FIFO. If the FIFO is four messages deep (FSIZE = 3), FIFOCIx will take on a value of 0 to 3, depending on the state of the FIFO.

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

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

Register 3-34: CxFIFOSTAx: CAN FIFO Status Register x (x = 1 to 31) (Continued)

bit 2 **TFERFFIF:** Transmit/Receive FIFO Empty/Full Interrupt Flag bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Empty Interrupt Flag
1 = FIFO is empty
0 = FIFO is not empty, at least one message is queued to be transmitted
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Full Interrupt Flag
1 = FIFO is full
0 = FIFO is not full

bit 1 **TFHRFHIF:** Transmit/Receive FIFO Half Empty/Half Full Interrupt Flag bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Half Empty Interrupt Flag
1 = FIFO is ≤ half full
0 = FIFO is > half full
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Half Full Interrupt Flag
1 = FIFO is ≥ half full
0 = FIFO is < half full

bit 0 **TFNRFNIF:** Transmit/Receive FIFO Not Full/Not Empty Interrupt Flag bit
TXEN = 1 (FIFO configured as a transmit FIFO):
Transmit FIFO Not Full Interrupt Flag
1 = FIFO is not full
0 = FIFO is full
TXEN = 0 (FIFO configured as a receive FIFO):
Receive FIFO Not Empty Interrupt Flag
1 = FIFO is not empty, has at least one message
0 = FIFO is empty

Note 1: FIFOCI[4:0] gives a zero-indexed value to the message in the FIFO. If the FIFO is four messages deep (FSIZE = 3), FIFOCIx will take on a value of 0 to 3, depending on the state of the FIFO.
2: This bit is updated when a message completes (or aborts) or when the FIFO is reset.
3: This bit is reset on any read of this register or when the TXQ is reset.

Register 3-35: CxTEFCONL: CAN Transmit Event FIFO Control Register Low

U-0	U-0	U-0	U-0	U-0	S/HC-1	U-0	S/HC-0
—	—	—	—	—	FRESET	—	UINC
bit 15							bit 8

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	TEFTSEN(1)	—	TEFOVIE	TEFFIE	TEFHIE	TEFNEIE
bit 7							bit 0

Legend:	S = Settable bit	HC = Hardware Clearable 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 15-11 **Unimplemented:** Read as '0'

bit 10 **FRESET:** FIFO Reset bit
1 = FIFO will be reset when bit is set, cleared by hardware when FIFO is reset; the user should poll whether this bit is clear before taking any action
0 = No effect

bit 9 **Unimplemented:** Read as '0'

bit 8 **UINC:** Increment Tail bit
1 = When this bit is set, the FIFO tail will increment by a single message
0 = FIFO tail will not increment

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

bit 5 **TEFTSEN:** Transmit Event FIFO Timestamp Enable bit(1)
1 = Timestamps elements in TEF
0 = Does not timestamp elements in TEF

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

bit 3 **TEFOVIE:** Transmit Event FIFO Overflow Interrupt Enable bit
1 = Interrupt is enabled for overflow event
0 = Interrupt is disabled for overflow event

bit 2 **TEFFIE:** Transmit Event FIFO Full Interrupt Enable bit
1 = Interrupt is enabled for FIFO full
0 = Interrupt is disabled for FIFO full

bit 1 **TEFHIE:** Transmit Event FIFO Half Full Interrupt Enable bit
1 = Interrupt is enabled for FIFO half full
0 = Interrupt is disabled for FIFO half full

bit 0 **TEFNEIE:** Transmit Event FIFO Not Empty Interrupt Enable bit
1 = Interrupt is enabled for FIFO not empty
0 = Interrupt is disabled for FIFO not empty

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-36: CxTEFCONH: CAN Transmit Event FIFO Control Register High

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	FSIZE[4:0](1)				
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 **FSIZE[4:0]:** FIFO Size bits(1)
11111 = FIFO is 32 messages deep
...
00010 = FIFO is 3 messages deep
00001 = FIFO is 2 messages deep
00000 = FIFO is 1 message deep

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

Note 1: These bits can only be modified in Configuration mode (OPMOD[2:0] = 100).

Register 3-37: CxTEFSTA: CAN Transmit Event FIFO Status Register

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	U-0	HS/C	R-0	R-0	R-0
—	—	—	—	TEFOVIF	TEFFIF(1)	TEFHIF(1)	TEFNEIF(1)
bit 7							bit 0

Legend:	C = Clearable bit	HS = Hardware Settable 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 15-4 **Unimplemented:** Read as '0'

bit 3 **TEFOVIF:** Transmit Event FIFO Overflow Interrupt Flag bit
1 = Overflow event has occurred
0 = No overflow event has occurred

bit 2 **TEFFIF:** Transmit Event FIFO Full Interrupt Flag bit(1)
1 = FIFO is full
0 = FIFO is not full

bit 1 **TEFHIF:** Transmit Event FIFO Half Full Interrupt Flag bit(1)
1 = FIFO is ≥ half full
0 = FIFO is < half full

bit 0 **TEFNEIF:** Transmit Event FIFO Not Empty Interrupt Flag bit(1)
1 = FIFO is not empty
0 = FIFO is empty

Note 1: These bits are read-only and reflect the status of the FIFO.

Register 3-38: CxFIFOUAxL: CAN FIFO User Address Register x (x = 1 to 31) Low(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
FIFOUA[15:8]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
FIFOUA[7:0]							
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 15-0 **FIFOUA[15:0]:** FIFO User Address bits
TXEN = 1 (FIFO configured as a transmit buffer):
A read of this register will return the address where the next message is to be written (FIFO head).
TXEN = 0 (FIFO configured as a receive buffer):
A read of this register will return the address where the next message is to be read (FIFO tail).

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

Register 3-39: CxFIFOUAxH: CAN FIFO User Address Register x (x = 1 to 31) High(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
FIFOUA[31:24]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
FIFOUA[23:16]							
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 15-0 **FIFOUA[31:16]:** FIFO User Address bits
TXEN = 1 (FIFO configured as a transmit buffer):
A read of this register will return the address where the next message is to be written (FIFO head).
TXEN = 0 (FIFO configured as a receive buffer):
A read of this register will return the address where the next message is to be read (FIFO tail).

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

Register 3-40: CxTEFUAL: CAN Transmit Event FIFO User Address Register Low(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TEFUA[15:8]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TEFUA[7:0]							
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 15-0 **TEFUA[15:0]:** Transmit Event FIFO User Address bits
A read of this register will return the address where the next event is to be read (FIFO tail).

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

Register 3-41: CxTEFUAH: CAN Transmit Event FIFO User Address Register High(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TEFUA[31:24]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TEFUA[23:16]							
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 15-0 **TEFUA[31:16]:** Transmit Event FIFO User Address bits
A read of this register will return the address where the next event is to be read (FIFO tail).

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

Register 3-42: CxTXQUAL: CAN Transmit Queue User Address Register Low(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TXQUA[15:8]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TXQUA[7:0]							
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 15-0 **TXQUA[15:0]:** Transmit Queue User Address bits
A read of this register will return the address where the next message is to be written (TXQ head).

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

Register 3-43: CxTXQUAH: CAN Transmit Queue User Address Register High(1)

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TXQUA[31:24]							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
TXQUA[23:16]							
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 15-0 **TXQUA[31:16]:** TXQ User Address bits
A read of this register will return the address where the next message is to be written (TXQ head).

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

Register 3-44: CxTRECL: CAN Transmit/Receive Error Count Register Low

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:			
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 Counter bits

bit 7-0 **RERRCNT[7:0]:** Receive Error Counter bits

Register 3-45: CxTRECH: CAN Transmit/Receive Error Count Register High

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

U-0	U-0	R-1	R-0	R-0	R-0	R-0	R-0
—	—	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
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 15-6 **Unimplemented:** Read as '0'

bit 5 **TXBO:** Transmitter in Error Bus Off State bit (TERRCNT[7:0] > 255)
In Configuration mode, TXBO is set since the module is not on the bus.

bit 4 **TXBP:** Transmitter in Error Bus Passive State bit (TERRCNT[7:0] > 127)

bit 3 **RXBP:** Receiver in Error Bus Passive State bit (RERRCNT[7:0] > 127)

bit 2 **TXWARN:** Transmitter in Error Warning State bit (128 > TERRCNT[7:0] > 95)

bit 1 **RXWARN:** Receiver in Error Warning State bit (128 > RERRCNT[7:0] > 95)

bit 0 **EWARN:** Transmitter or Receiver is in Error Warning State bit

Register 3-46: CxBDIAG0L: CAN Bus Diagnostics Register 0 Low

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
NTERRCNT[7:0]							
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
NRERRCNT[7:0]							
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 **NTERRCNT[7:0]:** Nominal Bit Rate Transmit Error Counter bits

bit 7-0 **NRERRCNT[7:0]:** Nominal Bit Rate Receive Error Counter bits

Register 3-47: CxBDIAG0H: CAN Bus Diagnostics Register 0 High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
DTERRCNT[7:0]							
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
DRERRCNT[7:0]							
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 **DTERRCNT[7:0]:** Data Bit Rate Transmit Error Counter bits

bit 7-0 **DRERRCNT[7:0]:** Data Bit Rate Receive Error Counter bits

Register 3-48: CxBDIAG1L: CAN Bus Diagnostics Register 1 Low

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EFMSGCNT[15:8]							
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
EFMSGCNT[7:0]							
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 15-0 **EFMSGCNT[15:0]:** Error-Free Message Counter bits

Register 3-49: CxBDIAG1H: CAN Bus Diagnostics Register 1 High

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
DLCMM	ESI	DCRCERR	DSTUFERR	DFORMERR	—	DBIT1ERR	DBIT0ERR
bit 15							bit 8

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TXBOERR	—	NCRCERR	NSTUFERR	NFORMERR	NACKERR	NBIT1ERR	NBIT0ERR
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 **DLCMM:** DLC Mismatch bit
During a transmission or reception, the specified DLC is larger than the PLSIZEx of the FIFO element.

bit 14 **ESI:** ESI Flag of Received CAN FD Message Set bit

bit 13 **DCRCERR:** Same as for Nominal Bit Rate

bit 12 **DSTUFERR:** Same as for Nominal Bit Rate

bit 11 **DFORMERR:** Same as for Nominal Bit Rate

bit 10 **Unimplemented:** Read as '0'

bit 9 **DBIT1ERR:** Same as for Nominal Bit Rate

bit 8 **DBIT0ERR:** Same as for Nominal Bit Rate

bit 7 **TXBOERR:** Device Went to Bus Off bit (and auto-recovered)

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

bit 5 **NCRCERR:** Received Message with CRC Incorrect Checksum bit
The CRC checksum of a received message was incorrect. The CRC of an incoming message does not match with the CRC calculated from the received data.

bit 4 **NSTUFERR:** Received Message with Illegal Sequence bit
More than five equal bits in a sequence have occurred in a part of a received message where this is not allowed.

Register 3-49: CxBDIAG1H: CAN Bus Diagnostics Register 1 High (Continued)

bit 3 **NFORMERR:** Received Frame Fixed Format bit
A fixed format part of a received frame has the wrong format.

bit 2 **NACKERR:** Transmitted Message Not Acknowledged bit
Transmitted message was not Acknowledged.

bit 1 **NBIT1ERR:** Transmitted Message Recessive Level bit
During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.

bit 0 **NBIT0ERR:** Transmitted Message Dominant Level bit
During the transmission of a message (or Acknowledge bit, or active error flag or overload flag), the device wanted to send a dominant level (data or identifier bit of logical value '0'), but the monitored bus value was recessive. During bus off recovery, this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceeding bus off recovery sequence (indicating the bus is not stuck at dominant or continuously disturbed).

Register 3-50: CxFLTCONxL: CAN Filter Control Register x Low (x = 0 to 7; a = 0, 4, 8, 12, 16, 20, 24, 28; b = 1, 5, 9, 13, 17, 21, 25, 29)

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTENb	—	—	FbBP4	FbBP3	FbBP2	FbBP1	FbBP0
bit 15							bit 8

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTENa	—	—	FaBP4	FaBP3	FaBP2	FaBP1	FaBP0
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 **FLTENb:** Enable Filter b to Accept Messages bit
1 = Filter is enabled
0 = Filter is disabled

bit 14-13 **Unimplemented:** Read as '0'

bit 12-8 **FbBP[4:0]:** Pointer to FIFO When Filter b Hits bits
11111 = Message matching filter is stored in FIFO 31
11110 = Message matching filter is stored in FIFO 30
...
00010 = Message matching filter is stored in FIFO 2
00001 = Message matching filter is stored in FIFO 1
00000 = Reserved; FIFO 0 is the TX Queue and cannot receive messages

bit 7 **FLTENa:** Enable Filter a to Accept Messages bit
1 = Filter is enabled
0 = Filter is disabled

bit 6-5 **Unimplemented:** Read as '0'

bit 4-0 **FaBP[4:0]:** Pointer to FIFO When Filter a Hits bits
11111 = Message matching filter is stored in FIFO 31
11110 = Message matching filter is stored in FIFO 30
...
00010 = Message matching filter is stored in FIFO 2
00001 = Message matching filter is stored in FIFO 1
00000 = Reserved; FIFO 0 is the TX Queue and cannot receive messages

Register 3-51: CxFLTCONxH: CAN Filter Control Register x High (x = 0 to 7; c = 2, 6, 10, 14, 18, 22, 26, 30; d = 3, 7, 11, 15, 19, 23, 27, 31)

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTENd	—	—	FdBP4	FdBP3	FdBP2	FdBP1	FdBP0
bit 15							bit 8

R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
FLTENc	—	—	FcBP4	FcBP3	FcBP2	FcBP1	FcBP0
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 **FLTENd:** Enable Filter d to Accept Messages bit
1 = Filter is enabled
0 = Filter is disabled

bit 14-13 **Unimplemented:** Read as '0'

bit 12-8 **FdBP[4:0]:** Pointer to FIFO When Filter d Hits bits
11111 = Message matching filter is stored in FIFO 31
11110 = Message matching filter is stored in FIFO 30
...
00010 = Message matching filter is stored in FIFO 2
00001 = Message matching filter is stored in FIFO 1
00000 = Reserved; FIFO 0 is the TX Queue and cannot receive messages

bit 7 **FLTENc:** Enable Filter c to Accept Messages bit
1 = Filter is enabled
0 = Filter is disabled

bit 6-5 **Unimplemented:** Read as '0'

bit 4-0 **FcBP[4:0]:** Pointer to FIFO When Filter c Hits bits
11111 = Message matching filter is stored in FIFO 31
11110 = Message matching filter is stored in FIFO 30
...
00010 = Message matching filter is stored in FIFO 2
00001 = Message matching filter is stored in FIFO 1
00000 = Reserved; FIFO 0 is the TX Queue and cannot receive messages

Register 3-52: CxFLTOBJxL: CAN Filter Object Register x Low (x = 0 to 31)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EID4	EID3	EID2	EID1	EID0	SID10	SID9	SID8
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
SID7	SID6	SID5	SID4	SID3	SID2	SID1	SID0
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 15-11 **EID[4:0]:** Extended Identifier Filter bits
In DeviceNet™ mode, these are the filter bits for the first two data bytes.

bit 10-0 **SID[10:0]:** Standard Identifier Filter bits

Register 3-53: CxFLTOBJxH: CAN Filter Object Register x High (x = 0 to 31)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	EXIDE	SID11	EID17	EID16	EID15	EID14	EID13
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
EID12	EID11	EID10	EID9	EID8	EID7	EID6	EID5
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 **EXIDE:** Extended Identifier Enable bit
If MIDE = 1:
1 = Matches only messages with Extended Identifier addresses
0 = Matches only messages with Standard Identifier addresses

bit 13 **SID11:** Standard Identifier Filter bit

bit 12-0 **EID[17:5]:** Extended Identifier Filter bits
In DeviceNet™ mode, these are the filter bits for the first two data bytes.

Register 3-54: CxMASKxL: CAN Mask Register x Low (x = 0 to 31)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
MEID4	MEID3	MEID2	MEID1	MEID0	MSID10	MSID9	MSID8
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
MSID7	MSID6	MSID5	MSID4	MSID3	MSID2	MSID1	MSID0
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 15-11 **MEID[4:0]:** Extended Identifier Mask bits
In DeviceNet™ mode, these are the mask bits for the first two data bytes.

bit 10-0 **MSID[10:0]:** Standard Identifier Mask bits

Register 3-55: CxMASKxH: CAN Mask Register x High (x = 0 to 31)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	MIDE	MSID11	MEID17	MEID16	MEID15	MEID14	MEID13
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
MEID12	MEID11	MEID10	MEID9	MEID8	MEID7	MEID6	MEID5
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 **MIDE:** Identifier Receive Mode bit
1 = Matches only message types (standard or extended address) that correspond to the EXIDE bit in the filter
0 = Matches 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 13 **MSID11:** Standard Identifier Mask bit

bit 12-0 **MEID[17:5]:** Extended Identifier Mask bits
In DeviceNet™ mode, these are the mask bits for the first two data bytes.

4.0 MODES OF OPERATION

The CAN FD Protocol Module has eight modes of operations:

- Configuration mode
- Normal CAN FD mode: Supports mixing of CAN FD and CAN 2.0 messages
- Normal CAN 2.0 mode: Will generate error frames while receiving CAN FD messages. The FDF bit is forced to zero and only CAN 2.0 frames are sent, even if the FDF bit is set in the transmit message object.
- Disable mode
- Listen Only mode
- Restricted Operation mode
- Internal Loopback mode
- External Loopback mode

The modes of operations can be grouped into four main groups: Configuration, Normal, Sleep and Debug (see Figure 4-1).

4.1 Mode Change

Figure 4-1 illustrates the possible mode transitions. New modes of operation are requested by writing to the REQOP[2:0] (CxCONH[10:8]) bits. The modes of operations do not change immediately. The modes will only change when the bus is Idle.

The current operating mode is indicated in the OPMOD[2:0] (CxCONH[7:5]) bits. The application can enable an interrupt on an OPMODx change or poll the OPMODx bits.

4.1.1 CHANGING BETWEEN NORMAL MODES

Directly changing between Normal modes is not allowed. The Configuration mode must be selected before a new Normal mode can be selected.

4.1.2 CHANGING BETWEEN DEBUG MODES

Directly changing between Debug modes is not allowed. The Configuration mode must be selected before a new Debug mode can be selected.

4.1.3 EXITING NORMAL MODE

The device will transition to Configuration or Sleep mode only after the current message is transmitted.

4.1.4 ENTERING AND EXITING DISABLE MODE

The CAN FD Protocol Module enters Disable mode after a Disable mode request. The device exits Disable mode after a mode request.

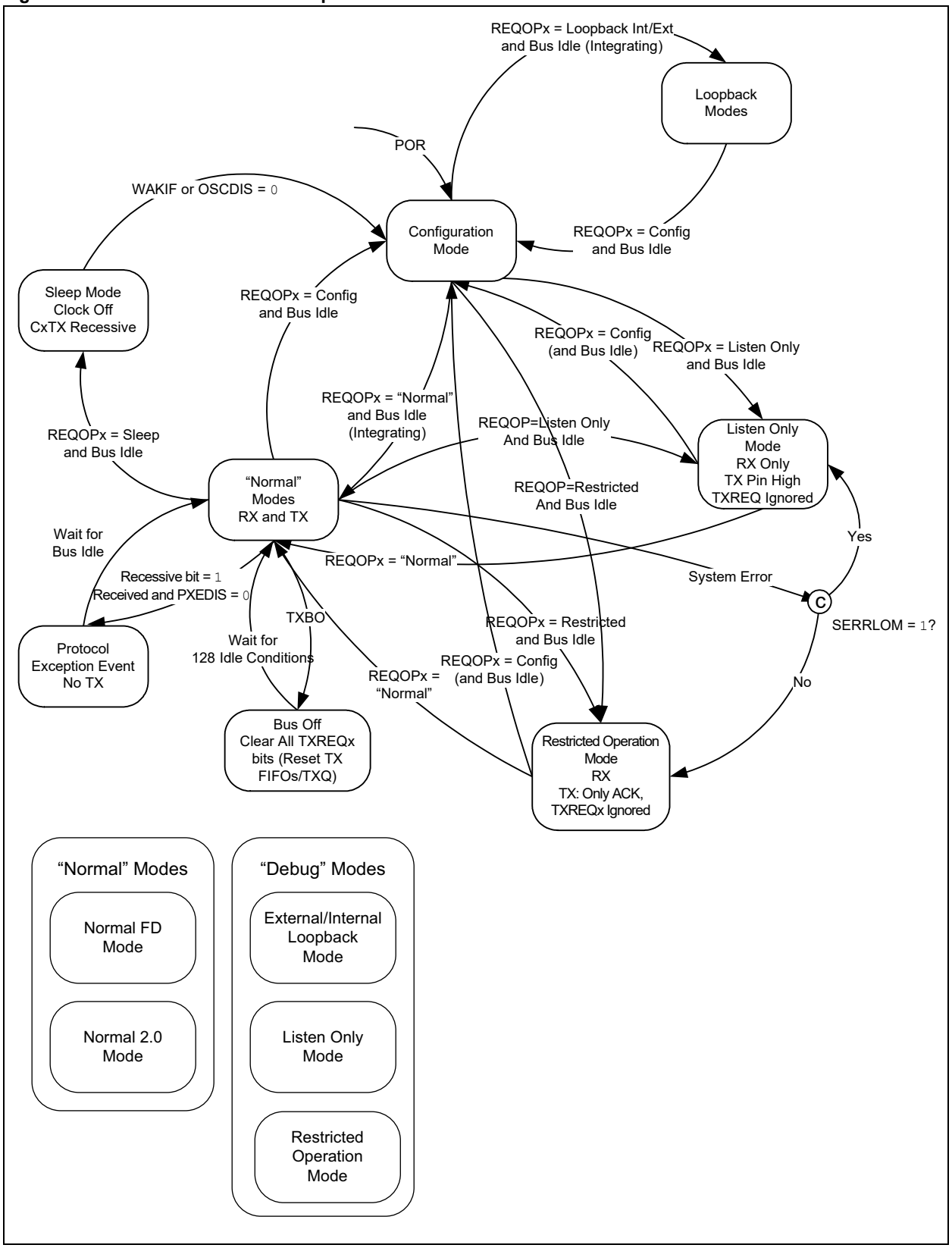
If WAKIE is set, a dominant edge on CxRX will generate an interrupt. The CPU has to enable the CAN module by requesting a Normal mode.

4.1.5 BUS INTEGRATING MODE

The CAN FD Protocol Module integrates to the bus, according to the ISO11898-1:2015 specifications (eleven consecutive recessive bits), under the following conditions:

- Change from Configuration mode to one of the Normal modes or Debug modes
- Change from Disable mode to one of the Normal modes

Figure 4-1: CAN FD Modes of Operation



4.2 Configuration Mode

After Reset, the CAN FD Protocol Module is in Configuration mode. The error counters are cleared and all registers contain the Reset values.

The CAN FD Protocol Module has to be initialized before activation. This is only possible when the module is in Configuration mode, OPMOD[2:0] = 100. The Configuration mode is requested by setting REQOP[2:0] = 100.

The CAN FD Protocol Module will protect the user from accidentally violating the CAN protocol through programming errors. The following registers and bit fields can only be programmed during Configuration mode:

- CxCONL: WAKFIL, CLKSEL, PXEDIS, ISOCRCEN
- CxCONH: TXQEN, STEF, SERRLOM, ESIGM, RTXAT
- CxNBTCFGL/H, CxDBTCFGL/H, CxTDCL/H
- CxTXQCONH: PLSIZE[2:0], FSIZE[4:0]
- CxFIFOCONxL: TXEN, RXTSEN
- CxFIFOCONxH: PLSIZE[2:0], FSIZE[4:0]
- CxTEFCONL: TEFTSEN
- CxTEFCONH: FSIZE[4:0]
- CxFIFOBAL/H

The CAN FD Protocol Module is not allowed to enter Configuration mode during transmission or reception to prevent the module from causing errors on the CAN bus. The following registers are reset when exiting Configuration mode:

- CxTRECL/H
- CxBDIAG0L/H
- CxBDIAG1L/H

In Configuration mode, FRESET is set in the CxFIFOCONxL, CxTXQCONL and CxTEFCONL registers, and all FIFOs and the TXQ are reset.

4.3 Normal Modes

4.3.1 NORMAL CAN FD MODE

Once the device is configured, Normal Operation mode can be requested by setting REQOP[2:0] = 000.

In this mode, the device will be on the CAN bus. It can transmit and receive messages in CAN FD mode, Bit Rate Switching can be enabled, and up to 64 data bytes can be transmitted and received.

4.3.2 NORMAL CAN 2.0 MODE

The Normal CAN 2.0 Operation mode can be requested by setting REQOP[2:0] = 110.

In this mode, the device will be on the CAN bus. This is a the Classic CAN 2.0 mode. The module will not receive CAN FD frames. It might send error frames if CAN FD frames are detected on the bus. The FDF, BRS and ESI bits in the TX objects will be ignored and transmitted as '0'.

4.4 Disable Mode

Disable mode is similar to Configuration mode, except the error counters are not reset. Disable mode is requested by setting REQOP[2:0] = 001.

The CAN module will not be allowed to enter Disable mode while a transmission or reception is taking place to prevent causing errors on the CAN bus. The module will enter Disable mode when the current message completes.

The OPMODx bits indicate whether the module successfully entered Disable mode. The application software should use this bit field as a handshake indication for the Disable mode request.

The CxTX pin will stay in the recessive state while the module is in Disable mode to prevent inadvertent CAN bus errors.

4.5 Debug Modes

4.5.1 LISTEN ONLY MODE

Listen Only mode is a variant of Normal CAN FD Operation mode. If the Listen Only mode is activated, the module on the CAN bus is passive. It will receive messages, but it will not transmit any bits. TXREQx bits will be ignored. No error flags or Acknowledge signals are sent. The error counters are deactivated in this state. The Listen Only mode can be used for detecting the baud rate on the CAN bus. It is necessary that there are at least two further nodes that communicate with each other. The baud rate can be detected empirically by testing different values until a message is received successfully. This mode is also useful for monitoring the CAN bus without influencing it.

4.5.2 RESTRICTED OPERATION MODE

In Restricted Operation mode, the node is able to receive data and remote frames, and to Acknowledge valid frames, but it does not send data frames, remote frames, error frames or overload frames. In case of an error or overload condition, it does not send dominant bits; instead, it waits for the bus to enter the Idle condition to resynchronize itself to the CAN communication. The error counters are not incremented.

4.5.3 LOOPBACK MODE

Loopback mode is a variant of Normal CAN FD Operation mode. This mode will allow internal transmission of messages from the transmit FIFOs to the receive FIFOs. The module does not require an external Acknowledge from the bus. No messages can be received from the bus, because the CxRX pin is disconnected.

4.5.3.1 Internal Loopback Mode

The transmit signal is internally connected to receive and the CxTX pin is driven high.

4.5.3.2 External Loopback Mode

The transmit signal is internally connected to receive and transmit messages, and can be monitored on the CxTX pin.

4.6 Low-Power Modes

4.6.1 SLEEP MODE

In the CAN module, special conditions need to be met for Sleep mode. The module must first be switched to Disable mode by setting REQOPx = 001. When OPMODx = 001, indicating Disable mode has been achieved, the CAN FD Protocol Module enters Sleep mode after a Sleep mode request.

In Sleep mode, the register contents do not change, so the OPMODx bits do not change. At the end of Sleep, the module will continue in the mode specified by the OPMODx bits previous to Sleep mode (which should be Disable mode, OPMODx = 001).

If the user executes a SLEEP instruction without switching to Disable mode, the module assumes a clock is available to read/write from RAM.

Since the system clock input is not available in Sleep mode, the CAN module cannot run as it requires a system clock to transmit or receive. Also, the FIFO is in system RAM, which has no clock in Sleep mode.

Recommended steps:

1. Write the REQOP[2:0] bits to '001'; the module will enter Disable mode.
2. Poll the OPMOD[2:0] bits to verify whether they are '001', which indicates that the module has successfully entered Disable mode.
3. Execute the SLEEP instruction.

4.6.2 IDLE MODE

The system can be set to run in a low-power mode, called Idle mode. When the device is in Idle mode, the CPU is disabled and only select peripherals are active.

Based on the configuration of the CAN SIDL bit, the module can either be in or out of Idle mode:

- If SIDL = 0, the module continues operation in Idle mode. If the module generates an interrupt while in Idle mode, the interrupt may generate a wake-up event.
- If SIDL = 1, the module stops when the device is in Idle mode. The module performs the same procedures when stopped in Idle mode as it does in Disable mode and the same requirements apply.

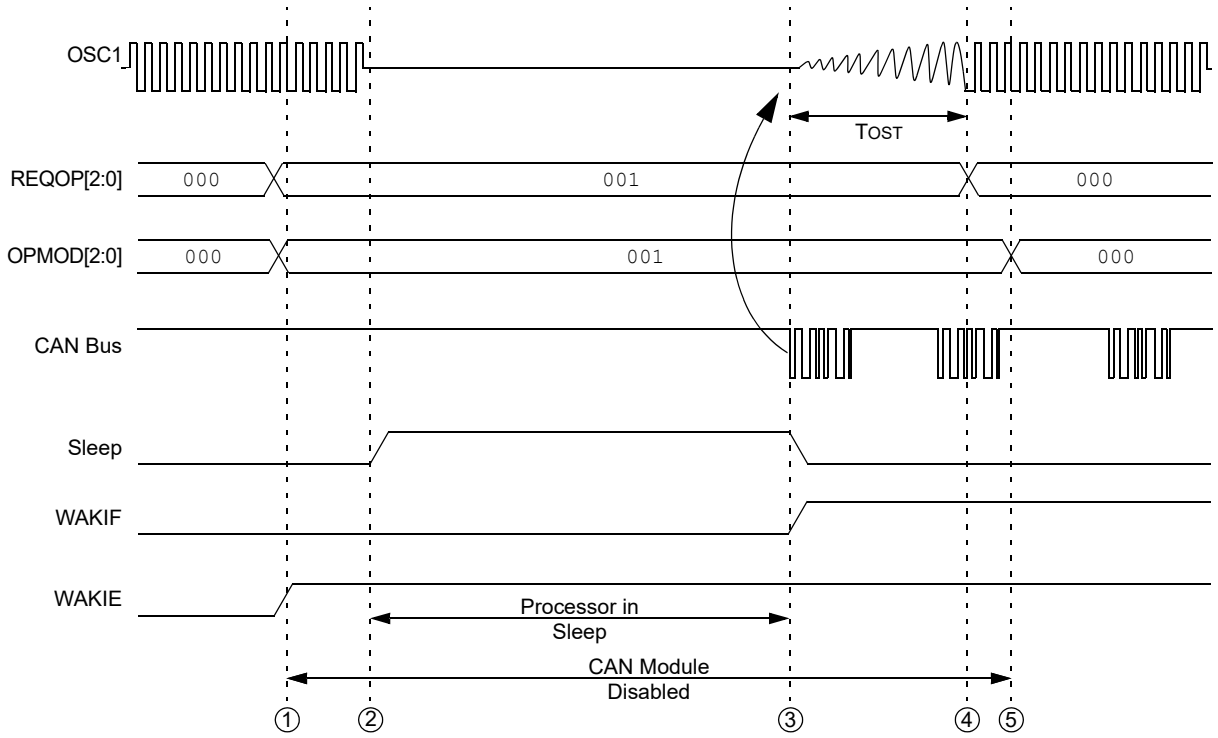
The user should ensure that the module is not active when the CPU transitions to Idle mode with SIDL = 1. To protect the CAN bus system from fatal consequences due to violation of this rule, the module will drive the TX pin into the recessive state while stopped in Idle mode.

If the CAN SIDL bit is set, the recommended procedure is to bring the module into Disable mode before the device is placed in Idle mode.

4.6.3 WAKE-UP FROM SLEEP

Figure 4-2 depicts how the CAN module will execute the SLEEP instruction and how the module wakes up on bus activity. Upon a wake-up from Sleep mode, the WAKIF flag is set.

Figure 4-2: Processor Sleep and CAN Bus Wake-up Interrupt



① – Processor requests and receives Module Disable mode. Wake-up interrupt is enabled.

② – Processor executes `SLEEP` (`PWRSAV #0`) instruction.

③ – SOF of message wakes up processor. Oscillator start time begins. CAN message is lost. WAKIF bit is set.

④ – Processor completes oscillator start time. Processor resumes program or interrupt, based on GIE bits. Processor requests Normal Operating mode. Module waits for 11 recessive bits before accepting CAN bus activity. CAN message is lost.

⑤ – Module detects 11 recessive bits. Module will begin to receive messages and transmits any pending messages.

The module will monitor the CAN receive line for activity while the module is Sleeping. The device will generate a wake-up interrupt on the falling edges of CxRX if WAKIE is enabled.

The device will exit Sleep mode after a new mode request or a negative edge on CxRX.

The module will be in Sleep mode if either of the following is true:

- The system is in Sleep mode following Disable mode
- The system is in Idle mode with SIDL = `1`

Note 1: If the module is in Sleep mode, the module generates an interrupt if the WAKIE bit (CxINTH[14]) is set and bus activity is detected. Due to delays in starting up the oscillator and CPU, the message activity that caused the wake-up will be lost.

2: The module can be programmed to apply a low-pass filter function to the CAN receive input line while in Disable, Sleep or Idle mode. This feature can be used to protect the module from wake-up due to short glitches on the CAN bus lines. The WAKFIL bit (CxCONL[8]) enables or disables the filter while the module is in Sleep.

5.0 CONFIGURATION

5.1 Clock Configuration

The sample point of all nodes in a CAN FD network should be at the same position. Hence, it is recommended to use the same clock frequency and bit time settings for all nodes. Therefore, a CAN clock (FCAN) of 80 MHz, 40 MHz or 20 MHz is recommended.

The CLKSEL bit allows the selection of the clock source to the CAN FD module.

- If CLKSEL = 1, then the auxiliary clock will be selected as a clock source
- If CLKSEL = 0, then the clock from the CAN clock generator will be selected

The following register is used to configure the CAN clock generator.

Register 5-1: CANCLKCON: CAN Clock Control Register(1)

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CANCLKEN	—	—	—	CANCLKSEL[3:0](1)			
bit 15							bit 8

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	CANCLKDIV[6:0](2,3)						
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 **CANCLKEN:** CAN Clock Generator Enable bit
1 = CAN clock generation circuitry is enabled
0 = CAN clock generation circuitry is disabled

bit 14-12 **Unimplemented:** Read as '0'

bit 11-8 **CANCLKSEL[3:0]:** Can Clock Source Select bits(1)
1011-1111 = Reserved (no clock selected)
1010 = AFVCO/4
1001 = AFVCO/3
1000 = AFVCO/2
0111 = AFVCO
0110 = AFPLLO
0101 = FVCO/4
0100 = FVCO/3
0011 = FVCO/2
0010 = FPLLO
0001 = FVCO
0000 = 0 (no clock selected)

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

Note 1: The user must ensure the input clock source is 640 MHz or less. Operation with input reference frequency above 640 MHz will result in unpredictable behavior.

2: The CANCLKDIVx divider value must not be changed during CAN module operation.

3: The user must ensure the maximum clock output frequency of the divider is 80 MHz or less.

4: Fvco/AFvco can be selected as CAN clock source in CANCLKSEL and used regardless of values of VCODIV/AVCODIV bits. Refer Oscillator FRM for the details of VCODIV/AVCODIV bits.

Register 5-1: CANCLKCON: CAN Clock Control Register(1)

bit 6-0 **CANCLKDIV[6:0]:** CAN Clock Divider Select bits(2,3)

1111111 = Divide by 128
• • •
0000010 = Divide by 3
0000001 = Divide by 2
0000000 = Divide by 1

Note 1: The user must ensure the input clock source is 640 MHz or less. Operation with input reference frequency above 640 MHz will result in unpredictable behavior.

2: The CANCLKDIVx divider value must not be changed during CAN module operation.

3: The user must ensure the maximum clock output frequency of the divider is 80 MHz or less.

4: Fvco/AFvco can be selected as CAN clock source in CANCLKSEL and used regardless of values of VCODIV/AVCODIV bits. Refer Oscillator FRM for the details of VCODIV/AVCODIV bits.

5.2 CAN Configuration

The CxCONL/H registers contain several bits that can only be configured in Configuration mode.

5.2.1 ISO CRC ENABLE

The module supports ISO CRC (according to ISO11898-1:2015) and non-ISO CRC (see **Section 2.1 "ISO vs. Non-ISO CRC"**). ISO CRC is enabled by setting the ISOCRCEN bit.

5.2.2 PROTOCOL EXCEPTION DISABLE

The negative edge between the FDF bit and the "reserved bit" in CAN FD frames is important for the calculation of the transceiver delay and for hard synchronization. Therefore, if the "reserved bit" following the FDF bit is detected recessive, the CAN FD Protocol Module will treat this as a form error. This is called, "Protocol Exception Event Detection Disabled," and is configured by setting the PXEDIS bit.

The Protocol Exception Event Detection Disabled can be enabled by clearing the PXEDIS bit. As a reaction to the protocol exception event, the error counters are not changed, hard synchronization is enabled, the module sends recessive bits and enters the bus integration state.

5.2.3 WAKE-UP FILTER – WFT[1:0]

The WAKFIL bit is used to enable/disable the low-pass filter on the CxRX pin. The filter is only active during Sleep mode. The WFTx bits allow the configuration of different filter times.

5.2.4 RESTRICTION OF TRANSMISSION ATTEMPTS

ISO11898-1:2015 requires that frames that lost arbitration and are not acknowledged, or are destroyed by errors, are automatically retransmitted. Optionally, the number of retransmission attempts can be limited.

When the RTXAT bit is set, retransmission attempts can be limited using the TXAT[1:0] bits in the FIFO Control registers. If the RTXAT bit is clear, then the TXATx bits in the FIFO Control register are ignored and the retransmission attempts are unlimited.

5.2.5 ERROR STATE INDICATOR (ESI) IN GATEWAY MODE

Normally, the ESI bit in a transmitted message reflects the error status of the CAN FD Protocol Module. ESI is transmitted recessive when the module is error passive. In case the module is used in a gateway application, there will be situations where the ESI bit in the message should be transmitted recessive, even though the gateway module is error active. This can be configured by setting the ESIGM bit.

5.2.6 MODE SELECTION IN CASE OF SYSTEM ERROR

The SERRLOM bit selects which mode the module will transition to in case of a system error. The module can either transition to Restricted Operation mode or Listen Only mode.

5.2.7 RESERVING MESSAGE MEMORY FOR TXQ AND TEF

Setting the TXQEN bit will reserve RAM for the TXQ. If the TXQEN bit is cleared, then the TXQ cannot be used.

Setting the STEF bit will reserve RAM for the TEF and all transmitted messages will be stored in the TEF.

5.3 CAN FD Bit Time Configuration

In order to achieve higher bandwidth, bits in a CAN FD frame are transmitted with two different bit rates:

- Nominal Bit Rate (NBR): Used during arbitration until the sample point of the BRS bit and the sample point of the CRC delimiter reach the EOF
- Data Bit Rate (DBR): Used during the data and CRC field

NBR is limited by the propagation delay of the CAN network (see **Section 5.3.2 "Propagation Delay"**). In the data phase, only one transmitter remains; therefore, the bit rate can be increased. The transmitting node always compares the intended transmitted bits with the actual bits on the CAN bus. The propagation delay in the data phase can be longer than the bit time. In this case, the data bits are sampled at a Secondary Sample Point (SSP) (see **Section 5.3.3 "Transmitter Delay Compensation (TDC)"**).

NBR is the number of bits per second during the arbitration phase. It is the inverse of the Nominal Bit Time (NBT) (see Equation 5-1).

Equation 5-1: Nominal Bit Rate/Time

$$NBR = \frac{1}{NBT}$$

DBR is the number of bits per second during the data phase. It is the inverse of the Data Bit Time (DBT) (see Equation 5-2).

Equation 5-2: Data Bit Rate/Time

$$DBR = \frac{1}{DBT}$$

The Baud Rate Prescaler (BRP) is used to divide the FCAN. The divided FCAN is used to generate the bit times.

There are two prescalers: NBRP for the Nominal Bit Rate Prescaler and DBRP for the Data Bit Rate Prescaler. The Time Quanta (NTQ and DTQ) are selected as shown in Equation 5-3 and Equation 5-4.

Equation 5-3: Nominal Time Quanta

$$NTQ = NBRP \times T_{CAN} = \frac{NBRP}{F_{CAN}}$$

Equation 5-4: Data Time Quanta

$$DTQ = DBRP \times T_{CAN} = \frac{DBRP}{F_{CAN}}$$

CAN bit times have four segments, as specified in ISO11898-1:2015 (see Figure 5-1).

Synchronization Segment (SYNC) – Synchronizes the different nodes connected on the CAN bus. A bit edge is expected to be within this segment. The Synchronization Segment is always 1 TQ.

Propagation Segment (PRSEG) – Compensates for the propagation delay on the bus. PRSEG has to be longer than the maximum propagation delay.

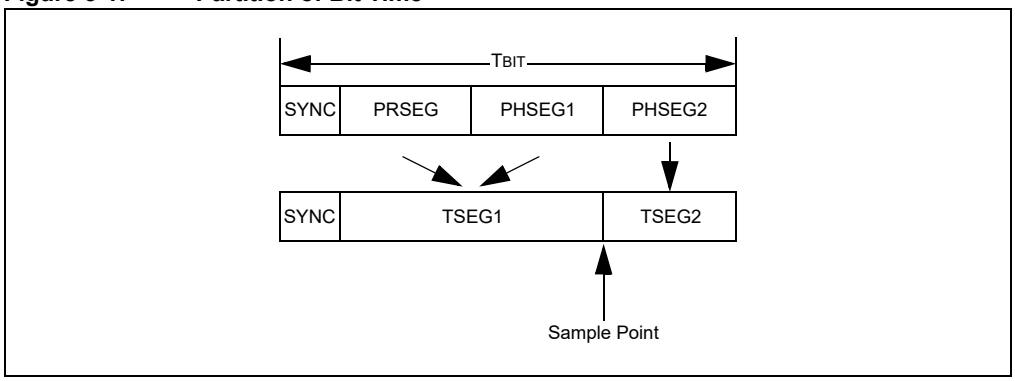
Phase Segment 1 (PHSEG1) – Compensates for errors that may occur due to phase shifts in the edges. The time segment may be automatically lengthened during resynchronization to compensate for the phase shift.

Phase Segment 2 (PHSEG2) – Compensates for errors that may occur due to phase shifts in the edges. The time segment may be automatically shortened during resynchronization to compensate for the phase shift.

In the bit time registers, PRSEG and PHSEG1 are combined to create TSEG1. PHSEG2 is called TSEG2. Each segment has multiple Time Quanta (TQ). The sample point lies between TSEG1 and TSEG2.

Table 5-1 and Table 5-2 show the ranges for the bit time configuration parameters.

Figure 5-1: Partition of Bit Time



The total number of TQ in a bit time is programmable and can be calculated using Equation 5-5 and Equation 5-6.

Equation 5-5: Number of NTQ in a NBT

$$\frac{NBT}{NTQ} = NSYNC + NTSEG1 + NTSEG2$$

Equation 5-6: Number of DTQ in a DBT

$$\frac{DBT}{DTQ} = DSYNC + DTSEG1 + DTSEG2$$

Table 5-1: Nominal Bit Rate Configuration Ranges

Segment	Minimum	Maximum
NSYNC	1	1
NTSEG1	2	256
NTSEG2	1	128
NSJW	1	128
NTQ per Bit	4	385

Table 5-2: Data Bit Rate Configuration Ranges

Segment	Minimum	Maximum
DSYNC	1	1
DTSEG1	1	32
DTSEG2	1	16
DSJW	1	16
DTQ per Bit	3	49

5.3.1 SAMPLE POINT

The sample point is the point in the bit time at which the logic level of the bit is read and interpreted. The sample point in percent can be calculated using Equation 5-7 and Equation 5-8.

Equation 5-7: Nominal Sample Point (%)

$$NSP = \frac{1 + NTSEG1}{\frac{NBT}{NTQ}} \times 100$$

Equation 5-8: Data Sample Point (%)

$$DSP = \frac{1 + DTSEG1}{\frac{DBT}{DTQ}} \times 100$$

5.3.2 PROPAGATION DELAY

Figure 5-2 illustrates the propagation delay between two CAN nodes on the bus, assuming Node A is transmitting a CAN message. The transmitted bit will propagate from the transmitting CAN Node A through the transmitting CAN transceiver, over the CAN bus, through the receiving CAN transceiver, into the receiving CAN Node B.

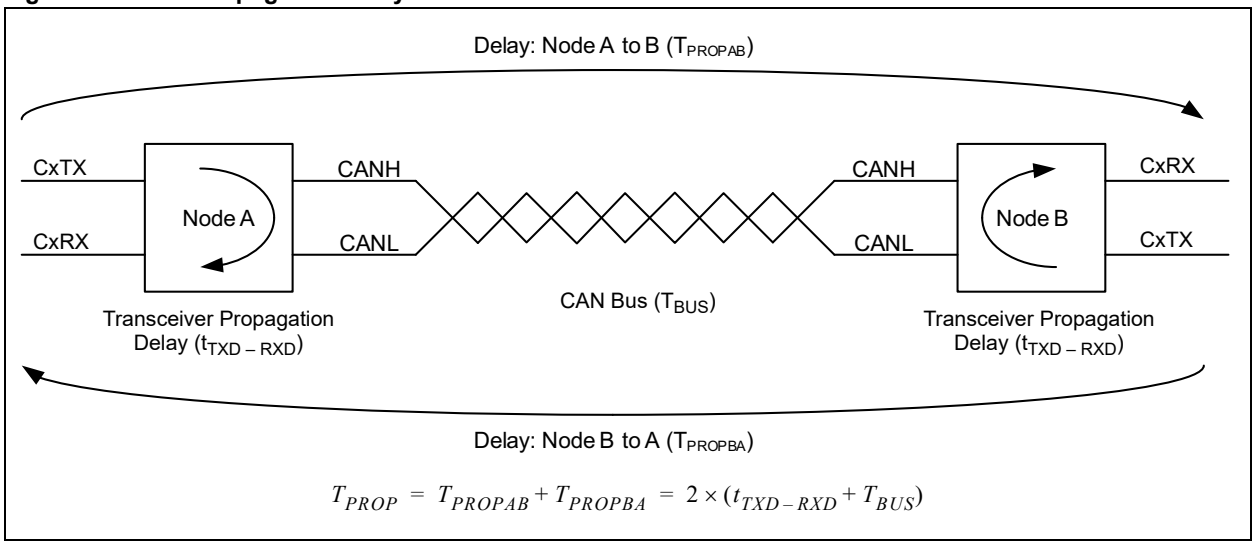
During the arbitration phase of a CAN message, the transmitter samples the CAN bus and checks if the transmitted bit matches the received bit. The transmitting node has to place the sample point after the maximum propagation delay.

Equation 5-9 describes the maximum propagation delay; where $t_{TXD-RXD}$ is the propagation delay of the transceiver, a maximum of 255 ns according to ISO11898-1:2015; T_{BUS} is the delay on the CAN bus, which is approximately 5 ns/m. The factor 2 comes from the worst-case when Node B starts transmitting exactly when the bit from Node A arrives.

Equation 5-9: Maximum Propagation Delay

$$T_{PROP} = 2 \times (t_{TXD-RXD} + T_{BUS})$$

Figure 5-2: Propagation Delay



5.3.3 TRANSMITTER DELAY COMPENSATION (TDC)

During the data phase of a CAN FD transmission, only one node is transmitting; the others are receiving. Therefore, the propagation delay does not limit the maximum data rate.

When transmitting via pin CxTX, the CAN FD Protocol Module receives the transmitted data from its local CAN transceiver via pin CxRX. The received data is delayed by the CAN transceiver's loop delay. In case this delay is greater than 1 + DTSEG1, a bit error would be detected.

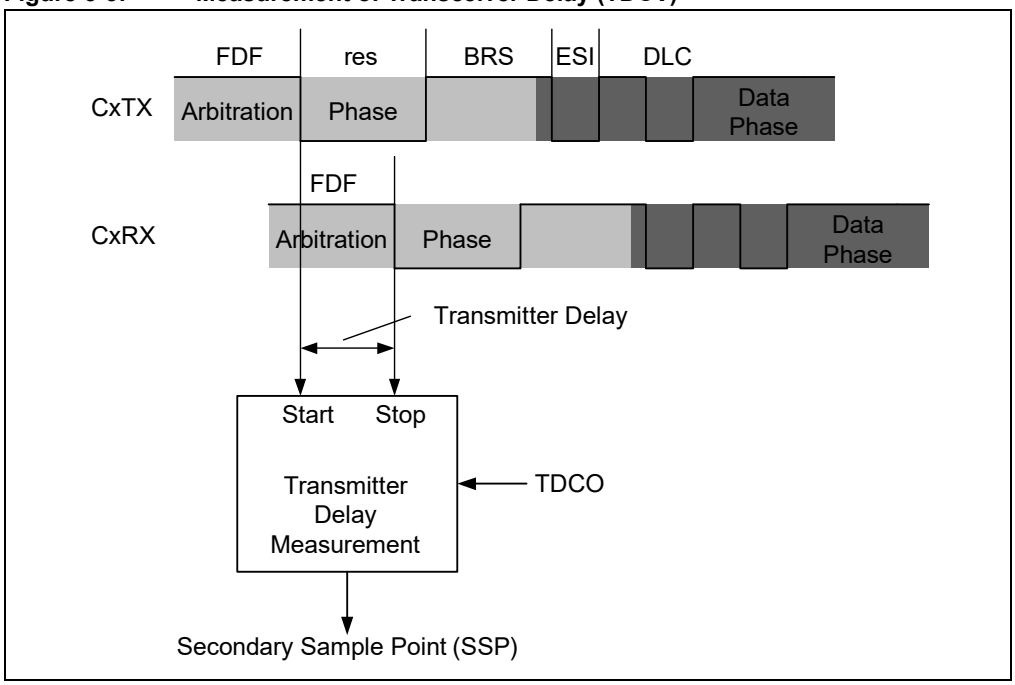
In order to enable a data phase bit time that is shorter than the transceiver loop delay, the Transmitter Delay Compensation (TDC) is implemented. Instead of sampling after DTSEG1, a Secondary Sample Point (SSP) is calculated and used for sampling during the data phase of a CAN FD message.

Figure 5-3 illustrates how the transceiver loop delay is measured and Equation 5-10 shows how the SSP is calculated.

Equation 5-10: Secondary Sample Point

$$SSP = TDCV[5:0] + TDCO[6:0]$$

Figure 5-3: Measurement of Transceiver Delay (TDCV)



5.3.4 SYNCHRONIZATION

To compensate for phase shifts between the oscillator frequencies of the nodes on the CAN bus, each CAN controller must be able to synchronize to the relevant edge of the incoming signal.

The CAN controller expects an edge in the received signal to occur within the SYNC segment. Only recessive-to-dominant edges are used for synchronization.

There are two mechanisms used for synchronization:

- **Hard Synchronization** – Forces the edge that has occurred to lie within the SYNC segment. The bit time counter is restarted with SYNC.
- **Resynchronization** – If the edge falls outside the SYNC segment, PHSEG1 or PHSEG2 will be adjusted.

For a more detailed description of the CAN synchronization, please refer to ISO11898-1:2015.

5.3.5 SYNCHRONIZATION JUMP WIDTH

The Synchronization Jump Width (SJW) is the maximum amount that PHSEG1 and PHSEG2 can be adjusted during resynchronization. SJW is programmable (see Table 5-1 and Table 5-2).

5.3.6 OSCILLATOR TOLERANCE

The oscillator tolerance, df, around the nominal frequency of the oscillator, $fnom$, is defined in Equation 5-11.

Equation 5-12 through Equation 5-16 describe the conditions for the maximum tolerance of the oscillator.

Equation 5-11: Oscillator Tolerance

$$(1 - df) \times fnom \leq F_{CAN} \leq (1 + df) \times fnom$$

Equation 5-12: Condition 1

$$df \leq \frac{NSJW}{2 \times 10 \times \frac{NBT}{NTQ}}$$

Equation 5-13: Condition 2

$$df \leq \frac{min(NPHSEG1, NPHSEG2)}{2 \times \left(13 \times \frac{NBT}{NTQ} - NPHSEG2\right)}$$

Equation 5-14: Condition 3

$$df \leq \frac{DSJW}{2 \times 10 \times \frac{DBT}{DTQ}}$$

Equation 5-15: Condition 4

$$df \leq \frac{min(NPHSEG1, NPHSEG2)}{2 \times \left(\left(6 \times \frac{DBT}{DTQ} - DPHSEG2\right) \times \frac{DBRP}{NBRP} + 7 \times \frac{NBT}{NTQ}\right)}$$

Equation 5-16: Condition 5

$$df \leq \frac{DSJW - max\left(0, \left(\frac{NBRP}{DBRP} - 1\right)\right)}{2 \times \left(\left(2 \times \frac{NBT}{NTQ} \times HNSEGP2\right) \times \frac{NBRP}{DBRP} + DPHSEG2 + 4 \times \frac{DBT}{DTQ}\right)}$$

5.3.7 RECOMMENDATIONS FOR BIT TIME CONFIGURATION

The following recommendations should be considered when configuring the bit time:

- Select the highest available CAN clock frequency:
  - Short TQ leads to high resolution to select the sample point
  - Use 20 MHz, 40 MHz or 80 MHz for FCAN
- Select the lowest NBRP and DBRP:
  - Low BRP leads to short TQ
  - NSYNC and DSYNC will be short and reduce the quantization error
  - The receiving node can synchronize more accurately to the transmitting node
- Set NBRP equal to DBRP:
  - Identical TQ in both phases prevents quantization errors during Bit Rate Switching
- Use the same Nominal Sample Point (NSP) and Data Sample Point (DSP) in all nodes on the CAN FD network:
  - Different sample points in the different nodes lead to different lengths of the BRS and CRC delimiter bits and introduce phase errors when switching the bit rate
  - NSP need not be equal to the DSP
  - The SSP can be different in differing CAN FD nodes
- Select the largest possible NSJW and DSJW:
  - Maximizes the oscillator tolerance
  - Allows the receiving nodes to quickly resynchronize to the transmitting nodes
- Enable automatic TDC for DBR of 1 Mbps and higher:
  - Automatic TDC measurement compensates for transmitter delay variations

5.3.8 BIT TIME CONFIGURATION EXAMPLE

The following tables illustrate the configuration of the CAN FD Bit Time registers, assuming there is a CAN FD network in an automobile with the following parameters:

- 500 kbps NBR – Sample Point at 80%
- 2 Mbps DBR – Sample Point at 80%
- 40 Meters – Minimum Bus Length

Table 5-3 and Table 5-4 illustrate how the bit time parameters are calculated. Since the parameters depend on multiple constraints and equations, and are calculated using an iterative process, it is recommended to enter the equations in a spreadsheet.

Table 5-5 translates the calculated values into register values. It is recommended to let the CAN FD Protocol Module measure the Transmitter Delay Compensation Value (TDCV). This is accomplished by setting TDCMOD[1:0] (CxTDCH[1:0]) = 10 (Automatic mode). In order to set the SSP to 80%, TDCO[6:0] are set to (DBRP * DTSEG1).

Table 5-3: Step-by-Step Nominal Bit Rate Configuration

Parameter	Constraint	Value	Unit	Equations and Comments
NBT	NBT ≥ 1 µs	2	µs	Equation 5-1.
FCAN	FCAN ≤ 80 MHz	80	MHz	CAN clock frequency= 80 MHz.
NBRP	1 to 256	1	—	Select smallest possible BRP value to maximize resolution.
NTQ	NBT, FCAN	12.5	ns	Equation 5-3.
NBT/NTQ	4 to 385	160	—	Equation 5-5.
NSYNC	Fixed	1	NTQ	Defined in ISO11898-1:2015.
NPRSEG	NPRSEG > T_{PROP}	95	NTQ	Equation 5-9: T_{PROP} = 910 ns, minimum NPRSEG = T_{PROP}/NTQ = 72.8 NTQ. Selecting 95 will allow up to a 60m bus length.
NTSEG1	2 to 256 NTQ	127	NTQ	Equation 5-7. Select NTSEG1 to achieve 80% NSP.
NTSEG2	1 to 128 NTQ	32	NTQ	There are 32 NTQ left to reach NBT/NTQ = 160.
NSJW	1 to 128 NTQ; SJW ≤ min (NPHSEG1, NPHSEG2)	32	NTQ	Maximizing NSJW lessens the requirement for the oscillator tolerance.

Table 5-4: Step-by-Step Data Bit Rate Configuration

Parameter	Constraint	Value	Unit	Equations and Comments
DBT	DBT ≥ 125 ns	500	ns	Equation 5-2.
DBRP	1 to 256	1	—	Selecting the same prescaler as for NBT ensures that the TQ resolution does not change during the Bit Rate Switching.
DTQ	DBT, FCAN	12.5	ns	Equation 5-4.
DBT/DTQ	3 to 49	40	—	Equation 5-6.
DSYNC	Fixed	1	DTQ	Defined in ISO11898-1:2015.
DTSEG1	1 to 32 DTQ	31	DTQ	Equation 5-7. Select DTSEG1 to achieve 80% DSP.
DTSEG2	1 to 16 DTQ	8	DTQ	There are 8 DTQ left to reach DBT/DTQ = 40.
DSJW	1 to 16 DTQ; SJW ≤ min (DPHSEG1, DPHSEG2)	8	DTQ	Maximizing DSJW lessens the requirement for the oscillator tolerance.
Oscillator Tolerance Conditions 1-5	Minimum of Conditions 1-5	0.78	%	Equation 5-11 through Equation 5-16.

Table 5-5: Bit Time Register Initialization (500k/2M)

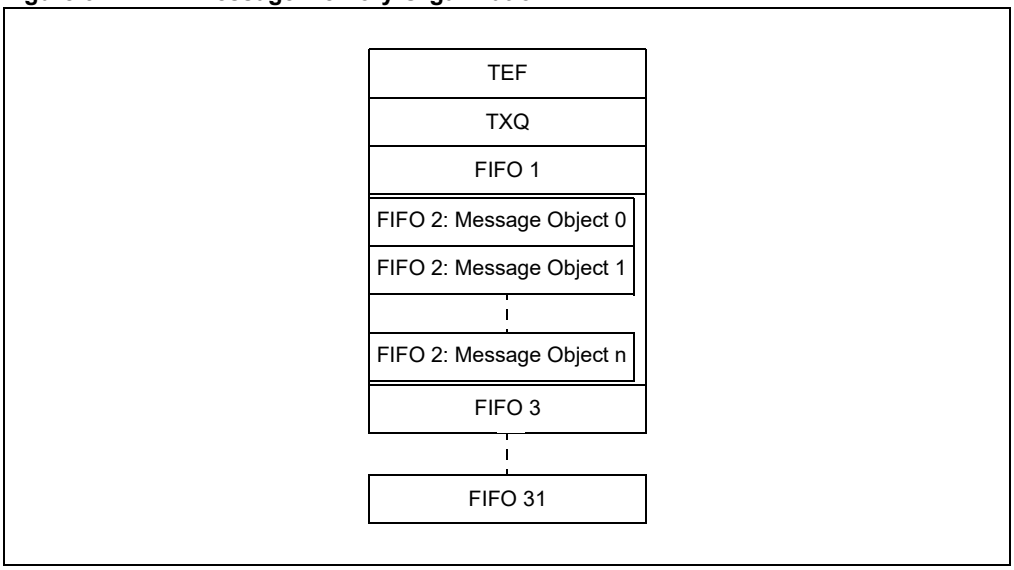
CxNBTCFGL/H	Value	CxDBTCFGL/H	Value	CxTDCL/H	Value
BRP[7:0]	0	BRP[7:0]	0	TDCMOD[1:0]	2
TSEG1[7:0]	126	TSEG1[4:0]	30	TDCO[6:0]	31
TSEG2[6:0]	31	TSEG2[3:0]	7	TDCV[5:0]	0
SJW[6:0]	31	SJW[3:0]	7	—	—

5.4 Message Memory Configuration

The message objects of the TEF, TXQ and transmit/receive FIFOs are located in RAM (see Figure 5-4). The application must configure the number of message objects in a FIFO between Message Object 0 and Message Object 31. Additionally, the application must configure the payload size of the message objects in each FIFO. This configuration determines where message objects are located in RAM. The RAM allocation can only be configured in Configuration mode. The start of the message memory is defined by CxFIFOBAL/H register, which is word aligned.

In order to optimize RAM usage, the application should start configuring the RAM with the TEF, followed by the TXQ, and continue with FIFO 1, FIFO 2, FIFO 3 and so on. In case a user application requires TEF, TXQ and 16 additional FIFOs, it should configure TEF and TXQ, followed by FIFO 1 through FIFO 16. It is not necessary to configure the unused FIFOs 17 through 31.

Figure 5-4: Message Memory Organization



5.4.1 TRANSMIT EVENT FIFO CONFIGURATION

In order to reserve space in RAM for the TEF, the STEF bit (CxCONH[3]) has to be set. The number of message objects in the TEF is configured using the FSIZE[4:0] bits (CxTEFCONH[12:8]). Transmitted messages can be timestamped by setting the TEFTSEN bit (CxTEFCONL[5]).

5.4.2 TRANSMIT QUEUE CONFIGURATION

In order to reserve space in RAM for the TXQ, the TXQEN bit (CxCONH[4]) has to be set. The number of message objects in the TXQ is configured using the FSIZE[4:0] bits (CxTXQCONH[12:8]. All objects in the TXQ use the same payload size (number of data bytes), which is configured using the PLSIZE[2:0] bits (CxTXQCONH[15:13]).

5.4.3 TRANSMIT FIFO CONFIGURATION

FIFO 1 through FIFO 31 can be configured as transmit FIFOs by setting TXEN in the CxFIFOCONxL register. The number of message objects in each transmit FIFO is configured using the FSIZE[4:0] bits (CxFIFOCONxH[12:8]). All objects in one transmit FIFO use the same payload size (number of data bytes), which is determined by the PLSIZE[2:0] bits (CxFIFOCONxH[15:13]).

5.4.4 RECEIVE FIFO CONFIGURATION

FIFO 1 through FIFO 31 can be configured as receive FIFOs by clearing TXEN in the CxFIFOCONxL register. The number of message objects in each receive FIFO is configured using the FSIZE[4:0] bits (CxFIFOCONxH[12:8]). All objects in one receive FIFO use the same payload size (number of data bytes), which is determined by the PLSIZE[2:0] bits (CxFIFOCONxH[15:13]). Received messages can be timestamped by setting the RXTSEN bit (CxFIFOCONxL[5]).

5.4.5 CALCULATION OF REQUIRED MESSAGE MEMORY

The size of required RAM depends on the configuration of each FIFO. Equation 5-17 through Equation 5-19 specify the sizes of the TEF, TXQ and the FIFOs in bytes. The TEF or TXQ is not used if their size is zero.

Since the size of the integrated RAM is limited, the user must check that the memory configuration fits into RAM. Equation 5-20 can be used to calculate the total RAM usage in bytes.

The size of the TEF objects depends on the enabling of timestamping. If TEFTSEN is set, then tefts = 4, else tefts = 0.

The $PayLoad(i)$ is defined in data bytes.

The size of a message object of an RX FIFO varies dependent on the enabling of timestamping. If RXTSEN = 1 and TXEN = 0 for $FIFO(i)$, then $rxts(i)$ = 4, else $rxts(i)$ = 0.

N is defined as the number of FIFOs used in addition to the TEF and the TXQ.

Equation 5-17: Size of TEF

$$S_{TEF} = N_{Elements}(TEF) \times (tefts + 8)$$

Equation 5-18: Size of TXQ

$$S_{TXQ} = N_{Elements}(TXQ) \times (8 + PayLoad(TXQ))$$

Equation 5-19: Size of FIFOs

$$S_{FIFO}(i) = N_{Elements}(i) \times (rxts(i) + 8 + PayLoad(i))$$

Equation 5-20: Total RAM Usage

$$S_{RAM} = \left(S_{TEF} + S_{TXQ} + \sum_{i=1}^{N} S_{FIFO}(i) \right)$$

For example:

- If TEF is 4 messages deep ($N_{Elements}$ (TEF) = 4) and TEFTSEN is clear, then the size of TEF = S_{TEF} = 4 x (0 + 8) = 32 bytes
- If $N_{Elements}$ (TXQ) = 1, PayLoad (TXQ) = 12, then the size of TXQ = S_{TXQ} = 1 x (8 + 12) = 20 bytes
- If $N_{Elements}$ (FIFO) = 3, PayLoad (FIFO) = 8, then the size of FIFO = S_{FIFO} = 3 x (8 + 8) = 48 bytes

Therefore, SRAM = S_{TEF} + S_{TXQ} + S_{FIFO} = 32 + 20 + 48 = 100 bytes.

5.4.6 CALCULATION OF START ADDRESSES

Since the payload size of the FIFOs can be configured individually, the start address of an individual object depends on the configuration of all previous objects. The application can read back the start addresses of each message object to double-check whether they were correctly configured.

The TEF starts at the beginning of the message memory.

Equation 5-21: Start Address of TEF

$$A_{TEF} = Base\ Address = CxFIFOBAL/H$$

The TXQ starts after the Transmit Event FIFO.

Equation 5-22: Start Address of TXQ

$$A_{TXQ} = CxFIFOBAL/H + S_{TEF}$$

The message FIFO object starts after the Transmit Queue.

Equation 5-23: Start Address of Message FIFO

$$A_{FIFO(1)} = CxFIFOBAL/H + S_{TEF} + S_{TXQ}$$

If CxCONH.STEF = 0, then TEF does not exist in RAM and STEF is zero. In this case, if TXQ is available, the TXQ starts at the base address defined by CxFIFOBAL/H.

If CxCONH.TXQEN = 0, then TXQ does not exist in RAM and STXQ is zero.

The start of nth FIFO can be calculated as follows:

Equation 5-24: Start Address of Nth FIFO

$$A_{FIFO(n)} = \left(A_{FIFO(1)} + \sum_{i=1}^{n-1} S_{FIFO}^{(i)} \right)$$

6.0 MESSAGE TRANSMISSION

The application has to configure the FIFO or TXQ before it can be used for transmission (see **Section 5.4.3 “Transmit FIFO Configuration”** and **Section 5.4.2 “Transmit Queue Configuration”**).

6.1 Transmit Message Object

Table 6-1 specifies the transmit message object used by the TXQ and the transmit FIFOs. The transmit objects contain the message ID, control bits and payload.

- **SID:** Standard Identifer or Base Identifier.
- **EID:** Extended Identifier.
- **DLC:** Data Length Code; specifies the number of data bytes to transmit (see **Section 2.1.1 “DLC Encoding”**).
- **IDE:** Identifier Extension; clearing this bit will transmit a base frame, setting this bit will transmit an extended frame.
- **RTR:** Remote Transmit Request; this bit is only specified in CAN 2.0 frames. Setting this bit will request a transmission of a receiving node.
- **FDF:** FD Format; if this bit is set, a CAN FD frame will be transmitted; otherwise, a CAN 2.0 frame will be transmitted. If Normal CAN 2.0 mode is selected, this bit is ignored and only CAN 2.0 frames are transmitted.
- **BRS:** Bit Rate Switch; the data phase of a CAN FD frame will be transmitted using DBR if this bit is set. If the bit is clear, the whole frame will be transmitted using NBR.
- **ESI:** Error State Indicator; normally, the ESI bit reflects the error status of the transmitting node. A recessive ESI bit in a CAN FD frame indicates that the transmitting node is error passive; a dominant bit shows that the transmitting node is error active. If ESIGM (CxCONH[1]) = 0, this bit in the object is ignored. If ESIGM = 1, the ESI bit in the transmitted message will be transmitted recessive if the CAN FD Protocol Module is error passive, or if the ESI bit in the message object is set. A gateway application would use it to signal that the ESI bit of the transmitting node is set.
- **SEQ:** Sequence Number; SEQ is not transmitted on the CAN bus. It is used to keep track of the transmitted messages. SEQ is stored in the TEF message object.
- **Transmit Buffer Data:** contains the payload of the message. Only the number of data bytes specified by the DLC are transmitted. Byte 0 is transmitted first, followed by 1, 2 and so on.

6.2 Loading Messages into Transmit FIFO

Before loading a message into the FIFO, the application must ensure that the FIFO is not full. There is space in the FIFO if TFNRFNIF (CxFIFOSTAx[0]) is set. Loading a message into a full FIFO can corrupt a message that is being transmitted.

The FIFO user address (CxFIFOUAxL/H) points to the RAM of the next transmit message object where the application should store the message. T0 of the transmit message object is loaded first, followed by T1, T2 and so on. The maximum number of data bytes is limited by the configured payload. Only the number of data bytes specified by the DLC have to be loaded.

After the message object is loaded into RAM, the FIFO needs to be incremented by setting the UINC bit (CxFIFOCONxL[8]). Doing so will cause the CAN FD Protocol Module to increment the head of the FIFO and update CxFIFOUAxL/H.

Now the message is ready for transmission and the next message can be loaded at the new address.

6.3 Loading Messages Into Transmit Queue

Loading transmit message objects into the TXQ works similarly to loading message objects into a transmit FIFO. The application must check the CxTXQSTA register to see if there is space in the TXQ. The CxTXQUAL/H registers should be used instead of the CxFIFOUAxL/H registers to calculate the address to load the message and set the UINC bit (CxTXQCONL[8]) to increment the head of the TXQ.

Table 6-1: Transmit Message Object (TXQ and TX FIFO)

Words	Bits	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
T0	15:8	EID[4:0]					SID[10:8]		
	7:0	SID[7:0]							
T1	15:8	—	—	SID11	EID[17:13]				
	7:0	EID[12:5]							
T2	15:8	SEQ[6:0]							ESI
	7:0	FDF	BRS	RTR	IDE	DLC[3:0]			
T3	15:8	SEQ[22:15]							
	7:0	SEQ[14:7]							
T4(1)	15:8	Transmit Data Byte 1							
	7:0	Transmit Data Byte 0							
T5(1)	15:8	Transmit Data Byte 3							
	7:0	Transmit Data Byte 2							
T6	15:8	Transmit Data Byte 5							
	7:0	Transmit Data Byte 4							
T7	15:8	Transmit Data Byte 7							
	7:0	Transmit Data Byte 6							
Ti-1	15:8	Transmit Data Byte n-3							
	7:0	Transmit Data Byte n-2							
Ti	15:8	Transmit Data Byte n							
	7:0	Transmit Data Byte n-1							

bit 15:11 (T0) **EID[4:0]:** Extended Identifier bits

bit 10-0 (T0) **SID[10:0]:** Standard Identifier bits

bit 15-14 (T1) **Unimplemented:** Read as '0'

bit 13 (T1) **SID11:** In FD mode, the Standard ID can be Extended to 12 bits Using r1 bit

bit 12-0 (T1) **EID[17:5]:** Extended Identifier bits

bit 15-9 (T2) **SEQ[6:0]:** Sequence to Keep Track of Transmitted Messages in Transmit Event FIFO bits

bit 8 (T2) **ESI:** Error Status Indicator bit

In CAN to CAN Gateway mode (ESIGM (CxCONH[1]) = 1), the transmitted ESI flag is a "logical OR" of ESI (T1) and the error passive state of the CAN controller.

In Normal mode, ESI indicates the error status:
1 = Transmitting node is error passive
0 = Transmitting node is error active

bit 7 (T2) **FDF:** FD Frame bit

Distinguishes between CAN and CAN FD formats.

bit 6 (T2) **BRS:** Bit Rate Switch bit

Selects if Data Bit Rate is switched.

bit 5 (T2) **RTR:** Remote Transmission Request bit (not used in CAN FD)

bit 4 (T2) **IDE:** Identifier Extension bit

Distinguishes between base and extended format.

bit 3-0 (T2) **DLC[3:0]:** Data Length Code bits

bit 15:0 (T3) **SEQ[22:7]:** Sequence to Keep Track of Transmitted Messages in Transmit Event FIFO bits

Note 1: Data Bytes 0-n: Payload size is configured individually in the PLSIZE[2:0] bits (CxFIFOCONxH[15:13]).

6.4 Requesting Transmission of Message in Transmit FIFO

After a message is loaded into a transmit FIFO, the message is ready for transmission. The application initiates the transmission of all messages in a FIFO by setting the TXREQ bit (CxFIFOCONxL[9]) or by setting the corresponding bit in the CxTXREQL/H registers. When all messages are transmitted, TXREQ gets cleared. The application can request transmission of multiple FIFOs and the TXQ simultaneously. The FIFO or TXQ with the highest priority will start transmitting first. Messages in a FIFO will be transmitted First-In-First-Out.

Messages can be loaded into a FIFO while the FIFO is transmitting messages. Since TXREQ is cleared by the FIFO automatically after the FIFO empties, UINC and TXREQ of the CxFIFOCONxL register must be set at the same time after appending a message. This ensures that all messages in the FIFO are transmitted, including the appended messages.

6.5 Requesting Transmission of Message in Transmit Queue

After a message is loaded into the TXQ, the message is ready for transmission. The application initiates the transmission of all messages in the queue by setting TXREQ (CxTXQCONL[9]). When all messages have been transmitted, TXREQ will be cleared. The application can request transmission of the TXQ and multiple FIFOs simultaneously. The TXQ or FIFO of the CxTXQCONL register with the highest priority will start transmitting first. Messages in the TXQ will be transmitted based on their ID. The message with the highest priority ID and the lowest ID value will be transmitted first.

Messages can be loaded into the TXQ while the TXQ is transmitting messages. Since TXREQ is cleared by the TXQ automatically after the TXQ empties, UINC and TXREQ of the CxTXQCONL register must be set at the same time after appending a message. This ensures that all messages in the TXQ are transmitted, including the appended messages.

6.6 CxTXREQ Register

The CxTXREQL and CxTXREQH registers contain the TXREQ[31:0] bits of the TXQ and of all the TX FIFOs. They have the following purposes:

- The user application can request transmission of the TXQ and/or one or more TX FIFOs by setting the corresponding bits in the CxTXREQL/H registers. Clearing a bit does NOT abort any transmissions.
- Reading the CxTXREQH and CxTXREQL registers gives information about which transmit FIFOs have transmissions pending.

CxTXREQL[0] is mapped to the TXQ, CxTXREQL[1] is mapped to TX FIFO 1, CxTXREQL[2] is mapped to TX FIFO 2 and so on. CxTXREQH[31] is mapped to TX FIFO 31.

6.7 Transmit Priority

The transmit priority of the FIFOs and TXQ needs to be configured using the TXPRIx bits (CxFIFOCONxH[4:0] and CxTXQCONH[4:0]).

Before transmitting a message, the priorities of the TXQ and the TX FIFOs queued for transmission are compared. The FIFO/TXQ with the highest priority will be transmitted first. For example, if transmit FIFO 1 has a higher priority setting than FIFO 3, all messages in FIFO 1 will be transmitted first. If multiple FIFOs have the same priority, the FIFO with the highest index is transmitted. For example, if FIFO 1 and FIFO 3 have the same priority setting, all messages in FIFO 3 will be transmitted first. If the TXQ and one or more FIFOs have the same priority, all messages in the TXQ will be transmitted first.

The transmit priority will be recalculated after every successful transmission of a single message.

6.7.1 TRANSMIT PRIORITY OF MESSAGES IN FIFO

In this method, the messages in a FIFO are transmitted First-In-First-Out.

6.7.2 TRANSMIT PRIORITY OF MESSAGES IN TXQ

Messages in the TXQ are transmitted based on the message ID. The message with the lowest message ID (highest priority) is transmitted first.

6.7.3 TRANSMIT PRIORITY BASED ON ID

The goal of transmitting CAN messages based on ID is to avoid “Inner Priority Inversion”. If a low-priority message is waiting to get transmitted due to bus traffic (arbitration), a higher priority message could be prevented from being transmitted. The TXQ solves this issue by reprioritizing the messages in the queue based on priority (ID).

6.8 Transmit Bandwidth Sharing

The bandwidth sharing feature works as follows:

- After a successful transmission of a message, the module will remain Idle for *n* arbitration bit times before the module attempts to transmit the next message; it suspends the next transmission.
- After the device has received a message, the module can transmit the next message as soon as the bus is Idle.

This allows other nodes on the bus to transmit their messages, even though they are of lower priority.

The number of arbitration bit times between transmissions can be configured using the TXBWS[3:0] bits (CxCONH[15:12]).

6.9 Retransmission Attempts

The number of retransmission attempts can be configured as follows:

- Retransmission attempts are disabled
- Three retransmission attempts
- Unlimited retransmissions

The retransmission attempts can be restricted by setting RTXAT (CxCONH[0]). The number of retransmission attempts can be configured individually for each transmit FIFO and the TXQ using TXAT[1:0] (CxFIFOCONxH[6:5] and CxTXQCONH[6:5], respectively).

In case RTXAT = 0, unlimited retransmission attempts will be used for all transmit FIFOs and the TXQ, and TXATx will be ignored.

6.9.1 RETRANSMISSION ATTEMPTS DISABLED

TXREQ will be cleared after the attempt to transmit the message. If the message is not successfully transmitted due to loss of arbitration or due to an error, TXATIF in the CxFIFOSTAx or CxTXQSTA register will be set.

6.9.2 THREE RETRANSMISSION ATTEMPTS

In case an error is detected during transmission, the CAN FD Protocol Module will decrement the number of remaining attempts and try to retransmit the message the next time the bus is Idle. In case arbitration is lost, the number of remaining attempts will not change. If all retransmission attempts are exhausted, TXREQ will be cleared and TXATIF in CxFIFOSTAx or CxTXQSTA will be set.

Before retransmitting the message, the transmit priority will be recalculated. The retransmission attempts will be reinitialized if a different TX FIFO or TXQ is selected for transmission, or if a message is received after the last transmission attempt.

6.9.3 UNLIMITED RETRANSMISSIONS

TXREQ will be cleared only after all messages in the TX FIFO or TXQ are successfully transmitted.

6.10 Aborting Transmission

A pending transmission can only be aborted before the transmission of the message starts, before the Start-of-Frame (SOF).

The transmission of a specific FIFO can be aborted by clearing TXREQ in the CAN Transmit Queue Control register; it cannot be aborted by clearing the bits in the CxTXREQL/H registers. Writing a '0' to one of the bits in the CxTXREQL/H registers will be ignored. The TXABT bit in the CAN FIFO Status x register will be set after a successful abortion. TXREQ will remain set until the message either aborts or is successfully transmitted.

Setting ABAT (CxCONH[11]) will abort all pending messages of all FIFOs. After all TXREQx bits are cleared, ABAT has to be cleared in order to be able to transmit new messages.

Clearing TXREQ for a transmit FIFO will attempt to abort all transmissions in the FIFO. If a message is successfully transmitted, the FIFO index will be updated as normal. If the message is successfully aborted, the FIFO index will not change.

The user can then use the FIFO Message Index bits, FIFOCI[4:0] (CxFIFOSTAx[12:8]), to identify messages that are transmitted. To reset the transmit FIFO index and erase all pending messages, the user can set the FRESET bit. The FIFO can then be loaded with new messages to be transmitted.

6.11 Remote Transmit Request – RTR

The CAN bus system has a method for allowing a master node to request data from another node. The master sends a message with the RTR bit set. The message contains no data, only an address to trigger a filter match.

Remote frames are only specified for CAN 2.0 frames; they are not supported in CAN FD frames.

The filter that is configured to respond to a Remote Transmit Request will point to a FIFO that is configured for transmission and RTREN has to be set.

Automatic remote data requests can be handled without MCU intervention. If a FIFO is properly configured, when a filter matches and points to the FIFO, the FIFO will be queued for transmission.

The FIFO must be configured as follows:

- Set TXEN to '1'.
- A filter must be enabled and loaded with a matching message identifier
- The Buffer Pointer for that filter must point to the TX FIFO. (Normally, a filter points to an RX FIFO.)
- RTREN bit must be set to '1' to enable RTR.
- The FIFO must be preloaded with at least one message to be sent.

When an RTR message is received, and it matches a filter pointing to a properly configured transmit FIFO, the TXREQ bit is set, queuing the object for transmission according to priorities.

A FIFO will only be transmitted if TXEN and RTREN are set, and if it is NOT empty. When a request for a remote transmission occurs while the FIFO is empty, the event will be treated as an overflow and the RXOVIF bit will be set.

6.12 Mismatch of DLC and Payload Size During Transmission

The PLSIZEx bits reserve a certain number of bytes in the transmit FIFO. The CAN FD Protocol Module handles mismatches between the DLC and payload size as follows:

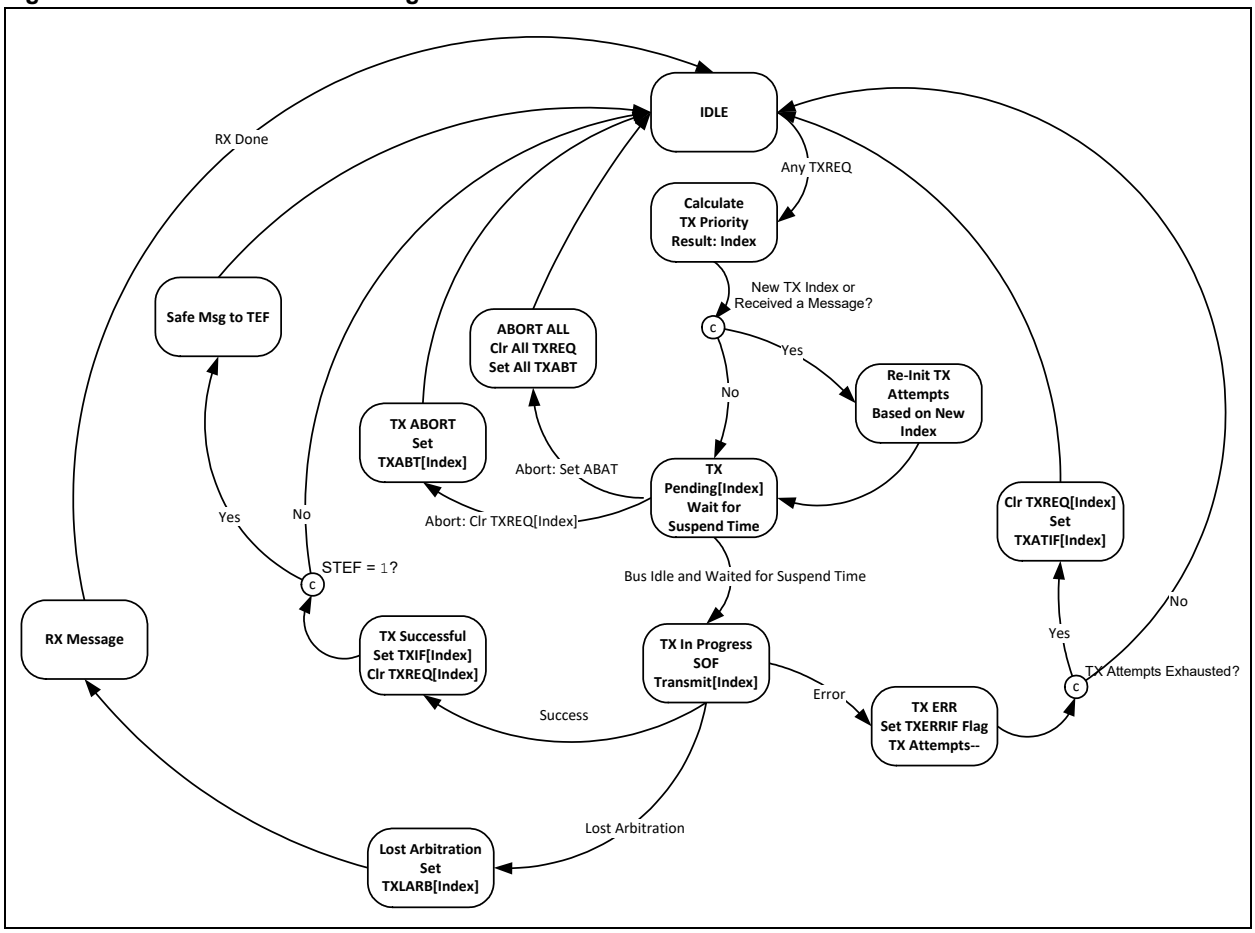
- If the DLC is smaller than the reserved payload, the number of data bytes specified by the DLC will be transmitted.
- If the DLC is bigger than the reserved payload, the module will not transmit the message. Instead, it will set the IVMIF (CxINTL[15]) and DLCMM (CxBDIAG1H[15]) flags and clear the TXREQ flag. The application can use the TEF to identify the message that is not transmitted.

6.13 Transmit State Diagram

Figure 6-1 describes how messages are queued for transmission. It illustrates how the most important transmit flags are set and cleared:

1. Messages are queued for transmission by setting the TXREQ flag.
2. The transmit priority will be determined. The FIFO or TXQ with the highest priority TXPRIx flag will be selected. The index of the TX message in the FIFO or TXQ will be calculated.
3. The TX message is pending for transmission.
4. Transmission can only start when the bus is Idle.
5. A pending transmission can only be aborted before SOF is transmitted.
6. During the transmission of a message, the CAN FD Protocol Module checks for the following:
   a) Loss of arbitration during the arbitration field.
   b) Transmit errors.
7. In case a message of a TX FIFO or the TXQ is transmitted successfully, the TXREQ will only be cleared after all messages of the FIFO are transmitted. After the transmission of any message, the status flags of the FIFO or TXQ are updated. In case STEF (CxCONH[3]) is set, the message will be stored into the TEF and a timestamp will be attached if enabled.
8. In case arbitration is lost, TXLARB of the TX FIFO or TXQ will be set and the device will switch over to receiving the message (see **Section 9.0 "Message Reception"**).
9. In case an error is detected during the transmission of a message, an error frame will be transmitted and the appropriate error flags will be set. Messages will be retransmitted according to **Section 6.9 "Retransmission Attempts"**.

Figure 6-1: Transmit State Diagram



6.14 Resetting Transmit FIFO

A Transmit FIFO can be reset by:

- Setting FRESET (CxFIFOCONxL[10]) or
- Placing the module in Configuration mode (OPMOD[2:0] = 100)

Resetting the FIFO will reset the Head and Tail Pointers, and the CxFIFOSTAx register. The settings in the CxFIFOCONxL and CxFIFOCONxH registers will not change.

Before resetting a TX FIFO using FRESET, ensure no transmissions are pending.

6.15 Resetting Transmit Queue

The Transmit Queue can be reset by:

- Setting FRESET (CxTXQCONL[10]) or
- Placing the module in Configuration mode (OPMOD[2:0] = 100)

Resetting the TXQ will reset the Head and Tail Pointers, and the CxTXQSTA register. The settings in the CxTXQCONL and CxTXQCONH registers will not change.

Before resetting the TXQ using FRESET, ensure no transmissions are pending.

6.16 Message Transmission Code Example

Example 6-1: Message Transmission Code

```
#include <xc.h>
/* This code example demonstrates a method to configure the CAN FD module to transmit Standard and
Extended ID CAN FD messages. This uses CAN1, TXQ and FIFO1. TXQ size is 1 and FIFO1 size is 2. */

/* Include fuse configuration code here. */

#define   MAX_WORDS 100
unsigned  int __attribute__((aligned(4)))CanTxBuffer[MAX_WORDS];

/*Data structure to implement a CANFD  message buffer. */
/* CANFD Message Time Stamp */
typedef   unsigned long CANFD_MSG_TIMESTAMP;

/* CAN TX Message Object Control*/
typedef   struct   _CANFD_TX_MSGOBJ_CTRL {
    unsigned DLC:4;
    unsigned IDE:1;
    unsigned RTR:1;
    unsigned BRS:1;
    unsigned FDF:1;
    unsigned ESI:1;
    unsigned SEQ:23;
    unsigned unimplemented1:16;
} CANFD_TX_MSGOBJ_CTRL;

/* CANFD TX Message ID*/
typedef   struct   _CANFD_MSGOBJ_ID {
    unsigned SID:11;
    unsigned long EID:18;
    unsigned SID11:1;
    unsigned unimplemented1:2;
} CANFD_MSGOBJ_ID;
```

Example 6-1: Message Transmission Code Example (Continued)

```
/* CAN TX Message Object*/
typedef   union    _CANFD_TX_MSGOBJ {
    struct  {
        CANFD_MSGOBJ_ID id;
        CANFD_TX_MSGOBJ_CTRL ctrl;
    } bF;
    unsigned int  word[4];
    unsigned char byte[8];
} CANFD_TX_MSGOBJ;

int main(void)
{
unsigned char index;

/* Place code to set device speed here. For this example the device speed should be set at 40 MHz
(i.e., the device is operating at 40 MIPS). */
ConfigureDeviceClockFor40MIPS();      // FCY = 40 MIPS

/* The dsPIC33C device features I/O remap. This I/O remap configuration for the CAN FD module can
be performed here. */
SetIORemapForCANFDModule();

/* Set up the CAN clock generator for 40 MHz and enable the CAN clock generator. */
ConfigureCANFDClockFor40MHz();       // FCAN = 40 MHz

/* Enable the CANFD module */
C1CONLbits.ON = 1;

/* Place CAN module in configuration mode */
C1CONHbits.REQOP = 4;
while(C1CONHbits.OPMOD != 4);

/* Initialize the C1FIFOBA with the start address of the CAN FIFO message buffer area. */
C1FIFOBAL = (unsigned int) &CanTxBuffer;

/* Set up the CANFD module for 1Mbps of Nominal bit rate speed and 2Mbps of Data bit rate. */
C1NBTCFGH = 0x001E;
C1NBTCFGL = 0x0707;
C1DBTCFGH = 0x000E;
C1DBTCFGL = 0x0303;
C1TDCH = 0x0002;                      //TDCMOD is Auto
C1TDCL = 0x0F00;

/* Configure CANFD module to enable Transmit Queue and BRS*/
C1CONLbits.BRSDIS = 0x0;
C1CONHbits.STEF = 0x0;                //Don't save transmitted messages in TEF
C1CONHbits.TXQEN = 0x1;

/* Configure TXQ to transmit 1 message*/
C1TXQCONHbits.FSIZE  = 0x0;           // single message
C1TXQCONHbits.PLSIZE = 0x7;           // 64 bytes of data

/* Configure FIFO1 to transmit 2 messages*/
C1FIFOCON1Hbits.FSIZE = 0x1;          //2 messages
C1FIFOCON1Hbits.PLSIZE = 0x2;         //16 bytes of data
C1FIFOCON1Lbits.TXEN = 0x1;           // Set TXEN bit ,transmit fifo
/* Place the CAN module in Normal mode. */
C1CONHbits.REQOP = 0;
while(C1CONHbits.OPMOD != 0);
```

Example 6-1: Message Transmission Code Example (Continued)

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

CANFD_TX_MSGOBJ *txObj;

/* Transmit message from TXQ - CANFD base frame with BRS*/

/* SID = 0x100, 64 bytes of data */
txObj  = (CANFD_TX_MSGOBJ *)C1TXQUAL;
txObj->bF.id.SID = 0x100;
txObj->bF.id.EID = 0x0000;
txObj->bF.ctrl.BRS = 1 ;              //Switch bit rate
txObj->bF.ctrl.DLC = 0xF;             //64 bytes
txObj->bF.ctrl.FDF = 1;               // CANFD frame
txObj->bF.ctrl.IDE = 0;               //Standard frame

for (index=0;index<0x40;index++ )
{
    txObj->byte[index+8] = 0x5A ;     // 64 bytes of 0x5A
}
C1TXQCONLbits.UINC = 1;               // Set UINC bit
C1TXQCONLbits.TXREQ = 1;              // Set TXREQ bit

/* Transmit message 0 from FIFO 1 - CANFD base frame with BRS*/

/* SID = 0x300 , 16 bytes of data */
txObj = (CANFD_TX_MSGOBJ *)C1FIFOUA1L;
txObj->bF.id.SID = 0x300;
txObj->bF.id.EID = 0x0000;
txObj->bF.ctrl.BRS = 1 ;              //Switch bit rate
txObj->bF.ctrl.DLC = 0xA;             //16 bytes
txObj->bF.ctrl.FDF = 1;               // CANFD frame
txObj->bF.ctrl.IDE = 0;               //Standard frame

for (index=0;index<0x10;index++ )
{
    txObj->byte[index+8] = 0xA5 ;     // 16 bytes of 0xA5
}
C1FIFOCON1Lbits.UINC  = 1;            // Set UINC bit
C1FIFOCON1Lbits.TXREQ = 1;            // Set TXREQ bit

/* Transmit message 1 from FIFO 1 - CANFD base frame with BRS*/

/* SID = 0x500, EID = 0xC000, 12 bytes of data */
txObj = (CANFD_TX_MSGOBJ *)C1FIFOUA1L;
txObj->bF.id.SID = 0x500;
txObj->bF.id.EID = 0xC000;
txObj->bF.ctrl.BRS = 1 ;              //Switch bit rate
txObj->bF.ctrl.DLC = 0x9;             //12 bytes
txObj->bF.ctrl.FDF = 1;               // CANFD frame
txObj->bF.ctrl.IDE = 1;               //Extended frame

for (index=0;index<0xC;index++ )
{
    txObj->byte[index+8] = 0x55 ;     // 12 bytes of 0x55
}
C1FIFOCON1Lbits.UINC  = 1;            // Set UINC bit
C1FIFOCON1Lbits.TXREQ = 1;            // Set TXREQ bit

while(1);
}
```

7.0 TRANSMIT EVENT FIFO – TEF

The TEF allows the application to keep track of the order and time in which the messages are transmitted. The TEF works similarly to a receive FIFO. Instead of storing received messages, it stores transmitted messages. Messages are only saved if STEF (CxCONH[3]) is set. The sequence number (SEQ) of the transmitted message is copied into the TEF object. The payload data is not stored. Transmitted messages are timestamped if TEFTSEN is set.

Table 7-1 specifies the TEF object. The first two words of the TEF object are a copy of the transmit message object. Optionally, the TEF object contains the timestamp when the message is transmitted.

7.1 Reading a TEF Object

Before reading a TEF object, the application must check that the TEF is not empty by reading the CxTEFSTA register. The TEF is not empty if TEFNEIF is set.

The TEF user address points to the address in RAM of the next TEF object to read. The actual address in RAM is calculated using Equation 7-1. TE0 of the TEF object is read first, followed by TE1 and TE2.

Equation 7-1: Start Address of TEF Object

$$A = BaseAddress = CxFIFOBAL/H$$

After the TEF object is read from RAM, the TEF needs to be incremented by setting UINC (CxTEFCONL[8]). This will cause the CAN FD Protocol Module to increment the Tail Pointer and update CxTEFUAL/H.

Now the next message can be read from the TEF.

7.1.1 RESETTING THE TEF

TEF can be reset by:

- Setting FRESET (CxTEFCONL[10]) or
- Placing the module in Configuration mode (OPMOD[2:0] = 100)

Resetting the FIFO will reset the Head and Tail Pointers, and the CxTEFSTA register. The settings in the CxTEFCONL and CxTEFCONH registers will not change.

Table 7-1: Transmit Event FIFO Object

Words	Bits	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
TE0	15:8	EID[4:0]					SID[10:8]		
	7:0	SID[7:0]							
TE1	15:8	—	—	SID11	EID[17:13]				
	7:0	EID[12:5]							
TE2	15:8	SEQ[6:0]							ESI
	7:0	FDF	BRS	RTR	IDE	DLC[3:0]			
TE3	15:8	SEQ[22:15]							
	7:0	SEQ[14:7]							
TE4(1)	15:8	TXMSGTS[15:8]							
	7:0	TXMSGTS[7:0]							
TE5(1)	15:8	TXMSGTS[31:24]							
	7:0	TXMSGTS[23:16]							

bit 15:11 (TE0) **EID[4:0]:** Extended Identifier bits

bit 10-0 (TE0) **SID[10:0]:** Standard Identifier bits

bit 15-14 (TE1) **Unimplemented:** Read as '0'

bit 13 (TE1) **SID11:** In FD mode, the Standard ID can be Extended to 12 Bits Using r1 bit

bit 12-0 (TE1) **EID[17:5]:** Extended Identifier bits

bit 15-9 (TE2) **SEQ[6:0]:** Sequence to Keep Track of Transmitted Messages in Transmit Event FIFO bits

bit 8 (TE2) **ESI:** Error Status Indicator bit

In CAN to CAN Gateway mode (ESIGM (CxCONH[1]) = 1), the transmitted ESI flag is a "logical OR" of ESI (TE2) and the error passive state of the CAN controller.

In Normal mode, ESI indicates the error status:
1 = Transmitting node is error passive
0 = Transmitting node is error active

bit 7 (TE2) **FDF:** FD Frame bit

Distinguishes between CAN and CAN FD formats.

bit 6 (TE2) **BRS:** Bit Rate Switch bit

Selects if Data Bit Rate is switched.

bit 5 (TE2) **RTR:** Remote Transmission Request bit (not used in CAN FD)

bit 4 (TE2) **IDE:** Identifier Extension bit

Distinguishes between base and extended format.

bit 3-0 (TE2) **DLC[3:0]:** Data Length Code bits

bit 15-0 (TE3) **SEQ[22:7]:** Sequence to Keep Track of Transmitted Messages in Transmit Event FIFO bits

bit 15-0 (TE4) **TXMSGTS[15:0]:** Transmit Message Timestamp bits

bit 15-0 (TE5) **TXMSGTS[31:16]:** Transmit Message Timestamp bits

Note 1: TE4 and TE5 (TXMSGTSx) only exist in objects where TEFTSEN (CxTEFCONL[5]) is set.

7.2 Transmit Event FIFO Code Example

A code example to save the transmitted messages using TEF is shown in Example 7-1.

Example 7-1: Using the Transmit Event FIFO Code

```
#include <xc.h>
/* This code example demonstrates a method to configure the CAN FD module to save the transmitted
messages in the TEF. This example uses CAN1, FIFO1 and TEF */

/* Include fuse configuration code here. */

#define  MAX_WORDS 100
unsigned int __attribute__((aligned(4)))CanTxBuffer[MAX_WORDS];  //message buffer to be written
unsigned int * currentMessageBuffer;                             // Points to message buffer to be read

/*data structure to implement a CANFD  message buffer. */
/* CANFD Message Time Stamp */
typedef  unsigned long CANFD_MSG_TIMESTAMP;
/* CAN TX Message Object Control*/
typedef  struct _CANFD_TX_MSGOBJ_CTRL {
    unsigned DLC:4;
    unsigned IDE:1;
    unsigned RTR:1;
    unsigned BRS:1;
    unsigned FDF:1;
    unsigned ESI:1;
    unsigned SEQ:23;
    unsigned unimplemented1:16;
} CANFD_TX_MSGOBJ_CTRL;
/* CANFD TX Message ID*/
typedef  struct _CANFD_MSGOBJ_ID {
    unsigned SID:11;
    unsigned long EID:18;
    unsigned SID11:1;
    unsigned unimplemented1:2;
} CANFD_MSGOBJ_ID;
/* CAN TX Message Object*/
typedef  union _CANFD_TX_MSGOBJ {
    struct  {
        CANFD_MSGOBJ_ID id;
        CANFD_TX_MSGOBJ_CTRL ctrl;
    } bF;
    unsigned int word[4];
    unsigned char byte[8];
} CANFD_TX_MSGOBJ;
/* CANFD TEF Message Object */
typedef  union _CAN_TEF_MSGOBJ {
    struct  {
        CANFD_MSGOBJ_ID id;
        CANFD_TX_MSGOBJ_CTRL ctrl;
        CANFD_MSG_TIMESTAMP timeStamp;
    } bF;
    unsigned int word[6];
    unsigned char byte[12];
} CANFD_TEF_MSGOBJ;
```

Example 7-1: Using the Transmit Event FIFO Code (Continued)

```
int main(void)
{
unsigned char index,fifoSize;

/* Place code to set device speed here. For this example the device speed should be set at
40 MHz (i.e., the device is operating at 40 MIPS). */
ConfigureDeviceClockFor40MIPS();                // FCY = 40 MIPS

/* The dsPIC33C device features I/O remap. This I/O remap configuration for the CAN FD
module can be performed here. */
SetIORemapForCANFDModule();

/* Set up the CAN clock generater for 40 MHz and enable the CAN clock generator. */
ConfigureCANFDClockFor40MHz();                  // FCAN = 40 MHz

/* Enable the CANFD module */
C1CONLbits.CON = 1;

/* Place CAN module in configuration mode */
C1CONHbits.REQOP = 4;
while(C1CONHbits.OPMOD != 4);
/* Initialize the C1FIFOBA with the start address of the CAN FIFO message buffer area. */
C1FIFOBAL = (unsigned int) &CanTxBuffer;

/* Set up the CANFD module for 1 Mbps of Nominal bit rate speed and 2Mbps of Data bit rate. */
C1NBTCFGH = 0x001E;
C1NBTCFGL = 0x0707;
C1DBTCFGH = 0x000E;
C1DBTCFGL = 0x0303;
C1TDCH = 0x0002;                                //TDCMOD is Auto
C1TDCL = 0x0F00;

/* Configure CANFD module to save transmitted messages in TEF and BRS*/
C1CONLbits.BRSDIS = 0x0;
C1CONHbits.STEF = 0x1;
C1CONHbits.TXQEN = 0x0;                         // Disable TXQ

/* Configure TEF to save 5 messages*/
C1TEFCONHbits.FSIZE = 0x4;                      // save 5 messages
C1TEFCONLbits.TEFTSEN = 1;

/* Configure FIFO1 to transmit 5 messages*/
C1FIFOCON1Hbits.FSIZE = 0x4;                    //5 messages
C1FIFOCON1Hbits.PLSIZE = 0x7;                   //64 bytes of data
C1FIFOCON1Lbits.TXEN = 0x1;                     // Set TXEN bit ,transmit fifo

/* Place the CAN module in Normal mode. */
C1CONHbits.REQOP = 0;
while(C1CONHbits.OPMOD != 0);

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

CANFD_TX_MSGOBJ *txObj;

/* Transmit 5 messages from FIFO 1 - CANFD base frame with BRS*/
/* SID = 0x300 , 64 bytes of data */
for      (fifoSize= 0; fifoSize < 5; fifoSize++)
{
```

Example 7-1: Using the Transmit Event FIFO Code (Continued)

```
    txObj = (CANFD_TX_MSGOBJ *)C1FIFOUA1L;
    txObj->bF.id.SID = 0x300;
    txObj->bF.id.EID = 0x0000;
    txObj->bF.ctrl.BRS = 1 ;                //Switch bit rate
    txObj->bF.ctrl.DLC = 0xF;               //64 bytes
    txObj->bF.ctrl.FDF = 1;                 //CANFD frame
    txObj->bF.ctrl.IDE = 0;                 //Standard frame
    txObj->bF.ctrl.SEQ = fifoSize ;         //Sequence does not get transmitted, but stored in TEF

    for(index=0;index<0x40;index++ )
    {
    txObj->byte[index+8] = 0xA5 ;           // 64 bytes of 0xA5
    }
    C1FIFOCON1Lbits.UINC = 1;                   // Set UINC bit
}

C1FIFOCON1Lbits.TXREQ = 1;                  // Set TXREQ bit
while    (C1FIFOCON1Lbits.TXREQ == 1);
/* Keep reading the TEF objects until the last transmitted message*/
for      (fifoSize= 0; fifoSize < 5; fifoSize++)
{
    while(C1TEFSTAbits.TEFNEIF ==0);
    CANFD_TEF_MSGOBJ *tefObj;
    tefObj = (CANFD_TEF_MSGOBJ *)C1TEFUAL;
    //ProcessTEFMessages (currentMessageBuffer) ;
    C1TEFCONLbits.UINC = 1 ;                    // Set UINC bit
}
while(1);
}
```

8.0 MESSAGE FILTERING

All messages on a CAN network will be received by all nodes. In order to process only messages of interest, a hardware filtering mechanism is implemented. The CAN FD Protocol Module can be configured to receive only messages of interest. The module contains a maximum of 32 acceptance filters. Each acceptance filter contains a filter object and a mask object. The user application configures the specific filter to receive a message with a given identifier by setting the filter object and mask object to match the identifier of the message to be received.

8.1 Filter Configuration

The filters are controlled by the CxFLTCONxL and CxFLTCONxH registers. The filters must be disabled by clearing the FLTEN bit before changing the filter or mask object; the module need not be in Configuration mode. After the filter object is updated, the Buffer Pointer, FnBP, has to be initialized and the filter can be enabled by setting the FLTEN bit. The FnBP points to the FIFO where the matching receive message needs to be stored.

8.2 Filtering a Received Message

The CAN FD Protocol Module starts acceptance filtering after the arbitration field and when the first three data bytes of a message are received. Figure 8-1 describes the flow of message filtering.

The module loops through all the filters, starting with Filter 0, which is the highest priority filter. The message in the Receive Message Assembly Buffer (RXMAB) is compared to the filter and mask. In case the message matches the filter and it is received without any errors, the message will be stored into the RX FIFO pointed to by the FnBP. Acceptance filtering is stopped and the associated RFIF bit is set.

In case an RTR is received, the TXREQ bit of the TX FIFO pointed to by FnBP will be set.

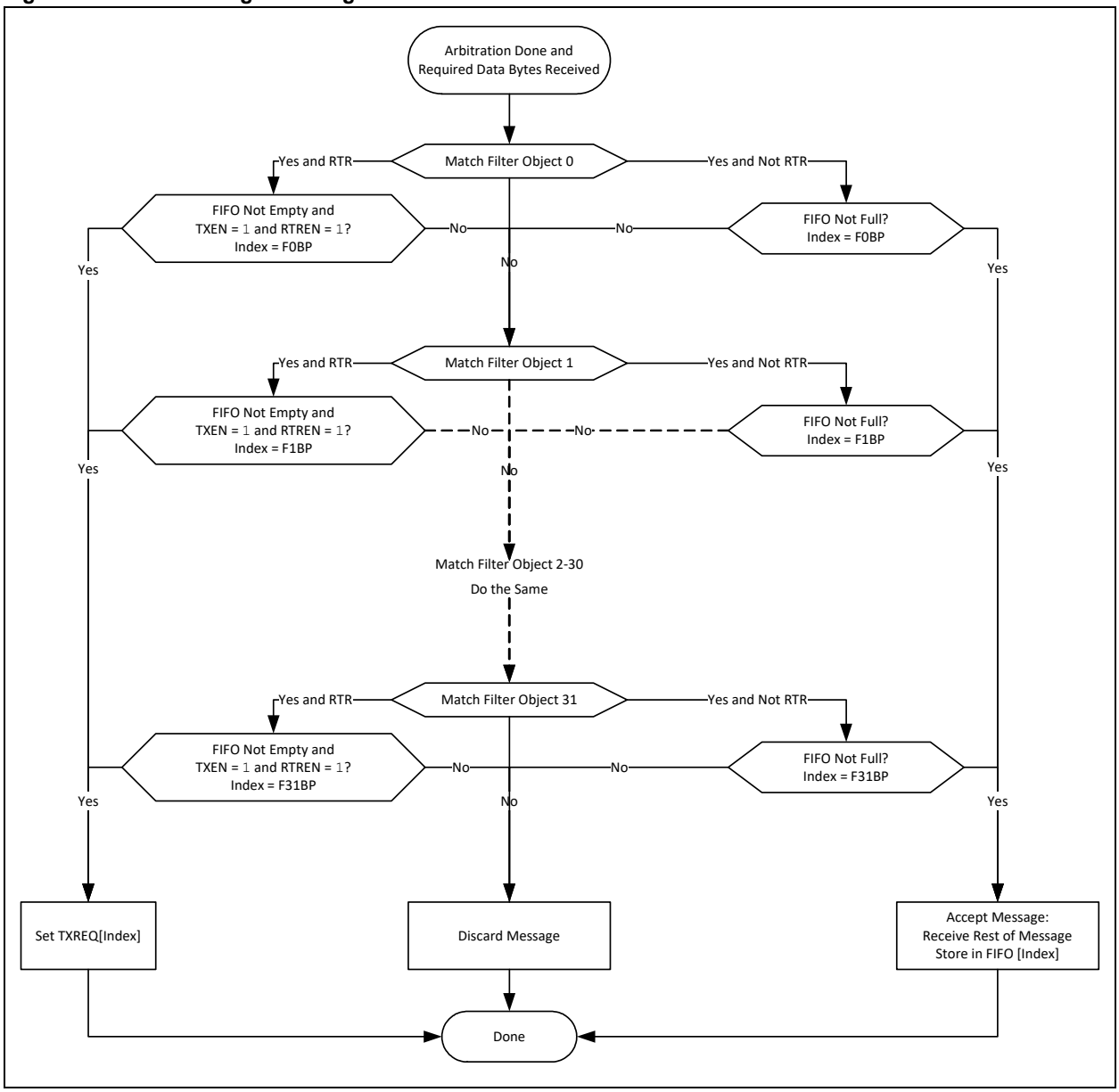
Filtering will continue with the next filter and RXOVIF will be set only when one of the following happens:

- A filter matches, but the RX FIFO is full.
- When multiple filters match the same message and all matching RX FIFOs are full, only the RXOVIF of the FIFO pointed to by the highest priority filter will be set.
- The RXOVIF bit will be set if the TX FIFO is empty during an RTR (TXEN = 1, RTREN = 1).

If none of the filters match, the received message will be discarded.

Note: If the module receives a message that matches a filter, but the corresponding FIFO is a TX FIFO (TXEN = 1, RTREN = 0), the module will discard the received message.

Figure 8-1: Message Filtering Flow



8.2.1 FILTERING STANDARD OR EXTENDED FRAMES

Figure 8-2 illustrates the flow of matching a single filter object to the received message in the RXMAB.

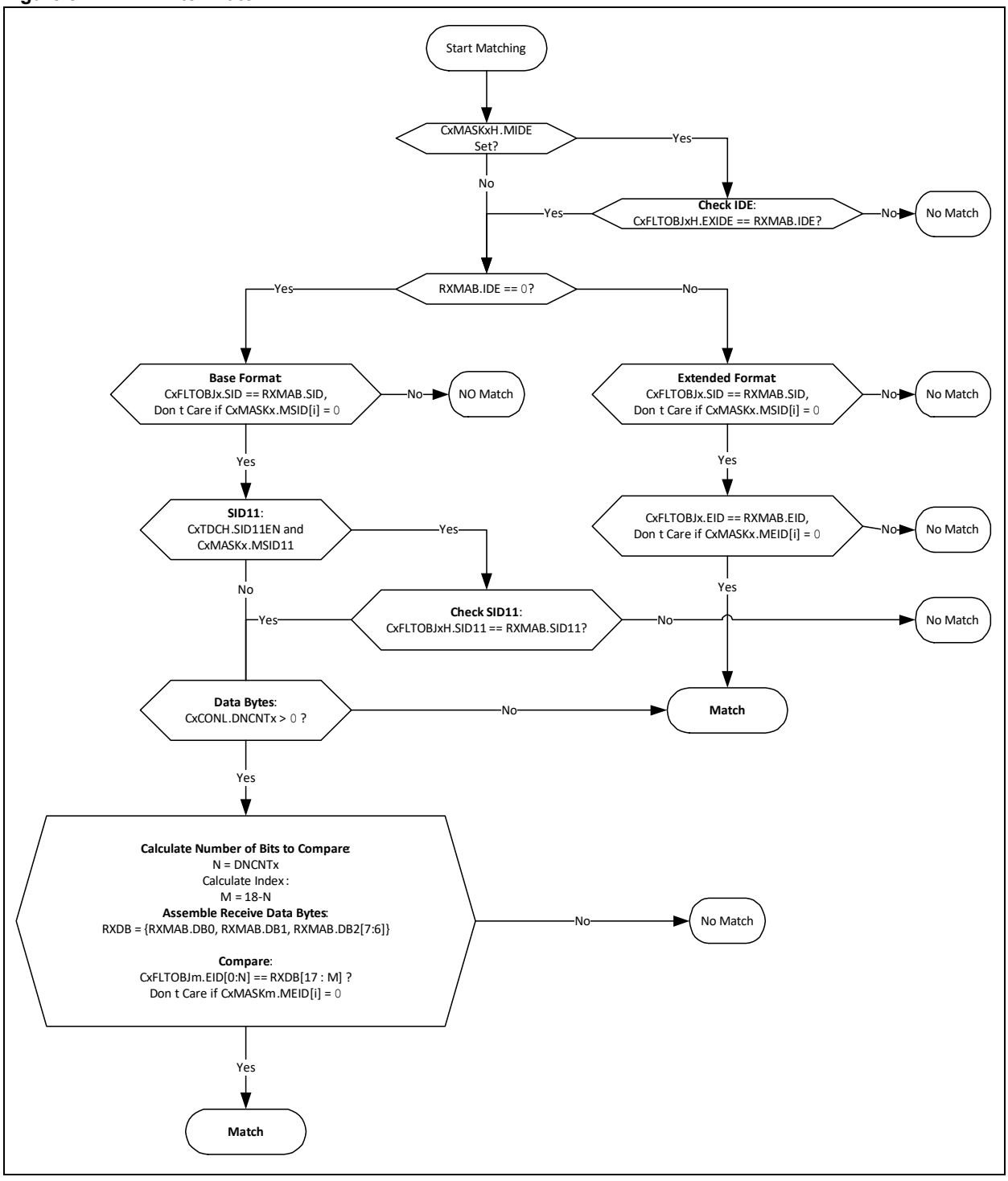
The filter object can be configured to accept either standard, extended or both frames. If MIDE is clear, both standard and extended frames will be accepted.

If the filter should only accept standard frames, then MIDE must be set and EXIDE must be cleared. If the filter should only accept extended frames, then both MIDE and EXIDE must be set.

8.2.2 MASK BITS

The mask object is used to ignore selected bits of the received identifier. The masked bits (mask bits with a value of '0') of the RXMAB will not be compared with the bits in the filter object. For example, to receive all messages with Identifiers 0, 1, 2 and 3, it is required to mask the lower two bits of the identifier by clearing the corresponding bits of the mask object.

Figure 8-2: Filter Match



8.2.3 FILTERING ON DATA BYTES

When the filter is configured to receive standard frames, the EID part of the filter and mask object can be selected to filter the data bytes. The DNCNT[4:0] bits in the CxCONL register are used to select how many bits in the data bytes are compared. Table 8-1 explains how many data bits are compared, and which filter bits and data bits are compared.

If DNCNTx is:

- '0', then data byte filtering is disabled.
- Non-zero, the filtering will commence on as many data bits as specified in DNCNTx. A filter hit will require matching of the SIDx bits and a match of *n* data bits with the filter's EID[0:17] bits. Data Byte 0[7] is always compared to EID[0], Data Byte 0[6] to EID[1], Data Byte 2[6] to EID[17].
- Greater than 18, indicating that the user-selected number of bits is greater than the total number of EIDx bits. The filter comparison will terminate with the 18th bit of the data.
- Greater than 16, and the received message has DLC = 2, indicating a payload of two data bytes. The filter comparison will terminate with the 16th bit of the data.
- Greater than 8, and the received message has DLC = 1, indicating a payload of one data byte. The filter comparison will terminate with the 8th bit of the data.
- Greater than 0, and the received message has DLC = 0, indicating no data payload. The filter comparison will terminate with the identifier.

8.2.4 12-BIT STANDARD ID

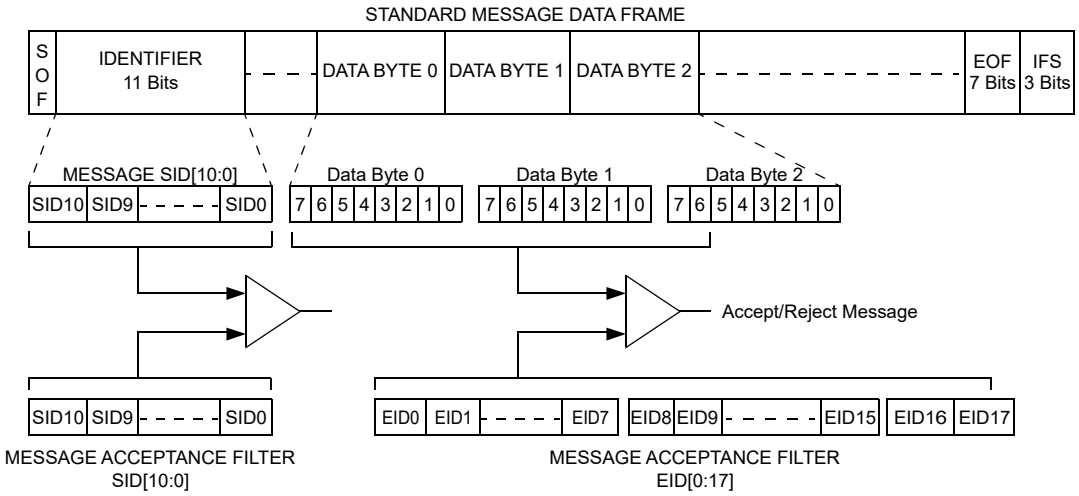
Setting SID11EN (CxTDCH[8]) allows the use of RRS as bit 12 of the SIDx (LSB). 12-Bit SID mode is only available for CAN FD base frames. The filter is extended by SID11 and MSID11. Data bytes can also be filtered in this mode.

Table 8-1: Data Byte Filter Configuration

DNCNT[4:0]	Received Message Data Bits to be Compared Byte [bits]	EIDx Bits Used for Acceptance Filter
00000	No Comparison	No Comparison
00001	Data Byte 0[7]	EID[0]
00010	Data Byte 0[7:6]	EID[0:1]
00011	Data Byte 0[7:5]	EID[0:2]
00100	Data Byte 0[7:4]	EID[0:3]
00101	Data Byte 0[7:3]	EID[0:4]
00110	Data Byte 0[7:2]	EID[0:5]
00111	Data Byte 0[7:1]	EID[0:6]
01000	Data Byte 0[7:0]	EID[0:7]
01001	Data Byte 0[7:0] and Data Byte 1[7]	EID[0:8]
01010	Data Byte 0[7:0] and Data Byte 1[7:6]	EID[0:9]
01011	Data Byte 0[7:0] and Data Byte 1[7:5]	EID[0:10]
01100	Data Byte 0[7:0] and Data Byte 1[7:4]	EID[0:11]
01101	Data Byte 0[7:0] and Data Byte 1[7:3]	EID[0:12]
01110	Data Byte 0[7:0] and Data Byte 1[7:2]	EID[0:13]
01111	Data Byte 0[7:0] and Data Byte 1[7:1]	EID[0:14]
10000	Data Byte 0[7:0] and Data Byte 1[7:0]	EID[0:15]
10001	Byte 0[7:0] and Byte 1[7:0] and Byte 2[7]	EID[0:16]
10010 to 11111	Byte 0[7:0] and Byte 1[7:0] and Byte 2[7:6]	EID[0:17]

Figure 8-3 illustrates how the first 18 data bits of the received message data payload are compared with the corresponding EIDx bits of the message acceptance filter (EID[17:0] bits in the CxFLTOBJxL/H registers). The IDE bit of the received message must be '0'.

Figure 8-3: CAN Operation with DeviceNet™ Filtering



Note: The DeviceNet™ filtering configuration shown for the EIDx bits is DNCNT[4:0] = 10010.

9.0 MESSAGE RECEPTION

The application has to configure the RX FIFO before it can be used for reception (see **Section 5.4.4 “Receive FIFO Configuration”**). In addition, the application has to configure and enable at least one filter (see **Section 8.1 “Filter Configuration”**).

The CAN FD Protocol Module continuously monitors the CAN bus. Messages that match a filter are stored in the RX FIFO pointed to by the filter (see **Section 8.2 “Filtering a Received Message”**). The message data is stored in the receive message objects.

9.1 Receive Message Object

Table 9-1 specifies the receive message object used by the RX FIFOs. The receive objects contain the message ID, control bits, payload and timestamp.

- **SID:** Standard Identifier (ID) or Base ID.
- **EID:** Extended Identifier.
- **DLC:** Data Length Code; specifies the number of data bytes in the frame (see **Section 2.1.1 “DLC Encoding”**).
- **IDE:** Identifier Extension; IDE = 0 means a Base Identifier frame is received. IDE = 1 means an Extended Identifier frame is received.
- **RTR:** Remote Transmit Request; this bit is only specified in CAN 2.0 frames. If this bit is set, the module is requested to respond with a frame transmission.
- **FDF:** FD Frame; if this bit is set, a CAN FD frame is received; otherwise, a CAN 2.0 frame is received.
- **BRS:** Bit Rate Switch; the data phase of a CAN FD frame is received using DBR if this bit is set. If the bit is clear, the whole frame is received using NBR.
- **ESI:** Error Status Indicator; the ESI bit reflects the error status of the transmitting node. A recessive ESI bit in a CAN FD frame indicates that the transmitting node is error passive; a dominant bit shows that the transmitting node is error active.
- **FILHIT:** Indicates the number of the filter that matched the received message.
- **RXMSGTS:** Timestamp of the Received Message; timestamping can be enabled for each RX FIFO individually using RXTSEN (CxFIFOCONxL[5]). The receive message object will not contain RXMSGTS if timestamping is disabled.
- **Receive Buffer Data:** Contains the payload of the message. The maximum payload is configured by the PLSIZEx bits (CxFIFOCONxH[15:13].

9.1.1 READING A RECEIVE MESSAGE OBJECT

Before reading a receive message object, the application must ensure that the RX FIFO is not empty by reading the CxFIFOSTAx register. The RX FIFO is not empty if TFNRFNIF is set.

The RX FIFO user address (CxFIFOUAxL/H) points to the RAM of the next receive message object to read. R0 of the receive message object is read first, followed by R1, R2 and so on.

After the receive message object is read from RAM, the RX FIFO needs to be incremented by setting the UINC bit (CxFIFOCONxL[8]). This will make the CAN FD Protocol Module increment to the tail of the FIFO and update CxFIFOUAxL/H.

Now the application can read the next message from the RX FIFO.

Table 9-1: Receive Message Object

Words	Bits	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
R0	15:8	EID[4:0]					SID[10:8]		
	7:0	SID[7:0]							
R1	15:8	—	—	SID11	EID[17:6]				
	7:0	EID[12:5]							
R2	15:8	FILHIT[4:0]					-—	-—	ESI
	7:0	FDF	BRS	RTR	IDE	DLC[3:0]			
R3	15:8	—	—	—	—	—	—	—	—
	7:0	—	—	—	—	—	—	—	—
R4(2)	15:8	RXMSGTS[15:8]							
	7:0	RXMSGTS[7:0]							
R5(2)	15:8	RXMSGTS[31:24]							
	7:0	RXMSGTS[23:16]							
R6(1)	15:8	Receive Data Byte 1							
	7:0	Receive Data Byte 0							
R7(1)	15:8	Receive Data Byte 3							
	7:0	Receive Data Byte 2							
R8	15:8	Receive Data Byte 5							
	7:0	Receive Data Byte 4							
R9	15:8	Receive Data Byte 7							
	7:0	Receive Data Byte 6							
Ri-1	15:8	Receive Data Byte n-2							
	7:0	Receive Data Byte n-3							
Ri	15:8	Receive Data Byte n							
	7:0	Receive Data Byte n-1							

bit 15-11 (R0) **EID[4:0]:** Extended Identifier bits
bit 10-0 (R0) **SID[10:0]:** Standard Identifier bits
bit 15-14 (R1) **Unimplemented:** Read as '0'
bit 13 (R1) **SID11:** In FD mode, the Standard ID can be Extended to 12 bits using r1 bit
bit 12-0 (R1) **EID[17:5]:** Extended Identifier bits
bit 15-11 (R2) **FILHIT[4:0]:** Filter Hit bits
The number of filters that matched.
bit 10-9 (R2) **Unimplemented:** Read as '0'
bit 8 (R2) **ESI:** Error Status Indicator bit
In CAN to CAN Gateway mode (ESIGM = 1), the transmitted ESI flag is a "logical OR" of ESI (T1) and the error passive state of the CAN controller.
In Normal mode, ESI indicates the error status:
1 = Transmitting node is error passive
0 = Transmitting node is error active
bit 7 (R2) **FDF:** FD Frame bit
Distinguishes between CAN and CAN FD formats.
bit 6 (R2) **BRS:** Bit Rate Switch bit
Selects if Data Bit Rate is switched.
bit 5 (R2) **RTR:** Remote Transmission Request bit (not used in CAN FD)
bit 4 (R2) **IDE:** Identifier Extension bit
Distinguishes between base and extended format.
bit 3-0 (R2) **DLC[3:0]:** Data Length Code bits
bit 15:0 (R3) **Unimplemented:** Read as '0'
bit 15:0 (R4) **RXMSGTS[15:0]:** Receive Message Timestamp bits
bit 15:0 (R5) **RXMSGTS[31:16]:** Receive Message Timestamp bits

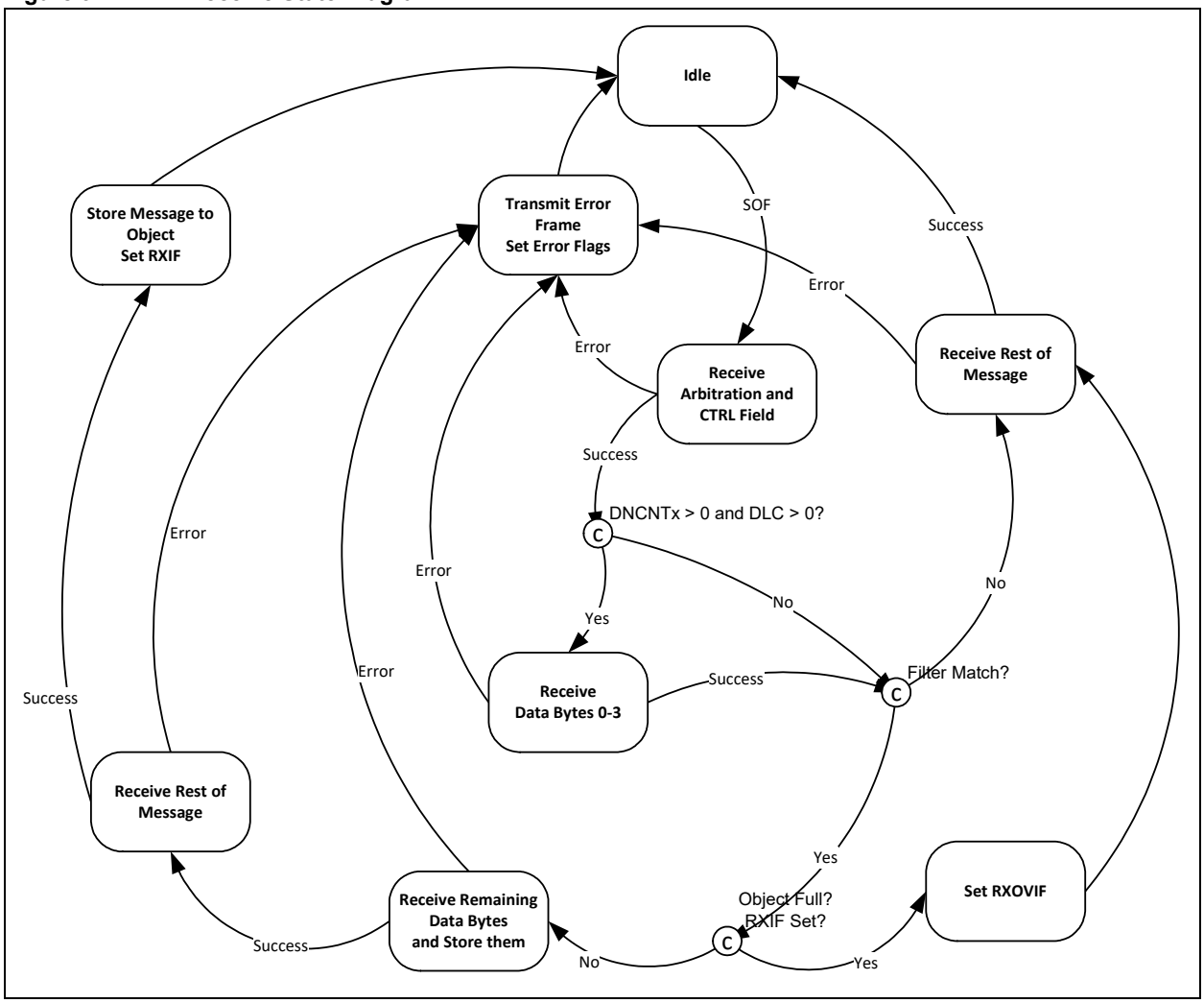
Note 1: Receive Message Object: Data Bytes 0-n; payload size is configured individually with the PLSIZE[2:0] bits.
2: R4 and R5 (RXMSGTSx) only exists in objects where RXTSEN is set.

9.2 Receive State Diagram

Figure 9-1 illustrates how messages are received. It illustrates how the most important receive flags are set and cleared.

- The CAN FD Protocol Module remains Idle until a SOF is detected.
- After a SOF is detected, the module will receive the arbitration and control fields.
- Based on the DNCNTx bits and the received DLC, acceptance filtering will start. See Figure 8-1 for more details.
- If none of the filters match, the message will still be received, but it will not be stored.
- If a filter matches, the device checks whether the receive object the filter points to is full.
- If the receive object is full, the RXOVIF bit will be set.
- If the receive object is not full, the rest of the data bytes are received and stored to the receive object.
- If a complete message is received, the message will be stored, a timestamp will be attached and the receive flags will be set; the FIFO status flags will be updated and the FIFO head will be incremented.
- In case an error is detected during the reception of a message, an error frame will be transmitted and the appropriate error flags will be set.

Figure 9-1: Receive State Diagram



9.3 Resetting RX FIFO

A receive FIFO can be reset by:

- Setting FRESET (CxFIFOCONxL[10]) or
- Placing the module in Configuration mode (OPMOD[2:0] = 100)

Resetting the FIFO will reset the Head and Tail Pointers, and the CxFIFOSTAx register. The settings in the CxFIFOCONxL/H registers will not change.

Before resetting an RX FIFO using FRESET, ensure that no enabled filter is pointing to the FIFO.

9.4 Mismatch of DLC and Payload Size During Reception

The PLSIZEx bits reserve a certain number of bytes in the receive message object. The module handles mismatches between DLC and payload size as follows:

- If the number of bytes specified by the DLC is smaller than the number of bytes specified by the PLSIZEx bits, the received message bytes will be stored in the message object, without any padding.
- If the number of bytes specified by the DLC is bigger than the number of bytes specified by the PLSIZEx bits, the data bytes that fit in the receive message object are stored and the other data bytes that do not fit are discarded. The module ensures that the next message object in RAM does not get overwritten. The module will store the message in the receive object and the RX FIFO status flags will be updated. In addition, the IVMIF (CxINTL[15]) and DLCMM flags (CxBDIAG1H[15]) will be set.

9.5 Message Reception Code Example

A code example to receive the CAN FD extended frame using Filter 0, and saving the messages in FIFO 1, is shown in Example 9-1.

Example 9-1: Message Reception Code

```
#include <xc.h>
/* This code example demonstrates a method to configure the CAN FD module to receive the extended ID
CAN FD messages. This uses CAN1, FIFO1 and filter 0. FIFO1 is configured to receive 2 messages. */

/* Include fuse configuration code here. */
#define  MAX_WORDS 100
unsigned int __attribute__((aligned(4)))CanRxBuffer[MAX_WORDS];

/*data structure to implement a CANFD message buffer. */
/* CANFD Message Time Stamp */
typedef  unsigned long CANFD_MSG_TIMESTAMP;

/* CANFD RX Message Object Control*/
typedef  struct _CANFD_RX_MSGOBJ_CTRL {
    unsigned DLC:4;
    unsigned IDE:1;
    unsigned RTR:1;
    unsigned BRS:1;
    unsigned FDF:1;
    unsigned ESI:1;
    unsigned unimplemented1:2;
    unsigned FilterHit:5;
    unsigned unimplemented2:16;
} CANFD_RX_MSGOBJ_CTRL;
```

Example 9-1: Message Reception Code (Continued)

```
/* CANFD RX Message ID*/
typedef  struct _CANFD_MSGOBJ_ID {
    unsigned SID:11;
    unsigned long EID:18;
    unsigned SID11:1;
    unsigned unimplemented1:2;
} CANFD_MSGOBJ_ID;

/* CANFD RX Message Object */
typedef  union _CANFD_RX_MSGOBJ {
    struct  {
        CANFD_MSGOBJ_ID id;
        CANFD_RX_MSGOBJ_CTRL ctrl;
        CANFD_MSG_TIMESTAMP timeStamp;
    } bF;
    unsigned int word[6];
    unsigned char byte [12]
} CANFD_RX_MSGOBJ;

int main(void)
{

/* Place code to set device speed here. For this example the device speed should be set at 40 MHz
(i.e., the device is operating at 40 MIPS). */
ConfigureDeviceClockFor40MIPS();             // FCY = 40 MIPS

/* The dsPIC33C device features I/O remap. This I/O remap configuration for the CAN FD module can
be performed here. */
SetIORemapForCANFDModule();

/* Set up the CAN clock generator for 40 MHz and enable the CAN clock generator. */
ConfigureCANFDClockFor40MHz();              // FCAN = 40 MHz

/* Enable the CANFD module */
C1CONLbits.CON = 1;

/* Place CAN module in configuration mode */
C1CONHbits.REQOP = 4;
while(C1CONHbits.OPMOD != 4);

/* Initialize the C1FIFOBA with the start address of the CAN FIFO message buffer area. */
C1FIFOBAL = (unsigned int) &CanRxBuffer;

/* Set up the CANFD module for 1 Mbps of Nominal bit rate speed and 2 Mbps of Data bit rate. */
C1NBTCFGH = 0x001E;
C1NBTCFGL = 0x0707;
C1DBTCFGH = 0x000E;
C1DBTCFGL = 0x0303;
C1TDCH = 0x0002;                            //TDCMOD is Auto
C1TDCL = 0x0F00;

/* Configure CANFD module to enable BRS */
C1CONLbits.BRSDIS = 0x0;
C1CONHbits.STEF = 0x0;                      //Don't save transmitted messages in TEF
C1CONHbits.TXQEN = 0x0;                     // No TXQ

/* Configure FIFO1 to Receive 2 messages*/
C1FIFOCON1Hbits.FSIZE = 0x1;                //2 messages
C1FIFOCON1Hbits.PLSIZE = 0x7;               //64 bytes of data
C1FIFOCON1Lbits.TXEN = 0x0;                 //Receive fifo
C1FIFOCON1Lbits.RXTSEN = 0x1;               //Enable receive fifo timestamp
```

Example 9-1: Message Reception Code (Continued)

```
/* Configure filter 0 and MASK 0 to accept extended id messages with id = 2 and 3 */
C1FLTCON0Lbits.F0BP = 1;                    // message stored in FIFO1
C1FLTOBJ0L = 0x1000;                        // EID = 0x00002
C1FLTOBJ0H = 0x4000;                        // Match messages with extended identifier address
C1MASK0L = 0xF7FF;                          // MEID = 0x1FFFE - Last it is 0
C1MASK0H = 0xFFFF;                          // Match message types
C1FLTCON0Lbits.FLTEN0 = 1;                  // Enable the filter 0

/* Place the CAN module in Normal mode. */
C1CONHbits.REQOP = 0;
while(C1CONHbits.OPMOD != 0);

/* Get the address of the message buffer to read the received messages.*/
/* set UINC bit to update the FIFO tail */
CANFD_RX_MSGOBJ *rxObj;
rxObj = (CANFD_RX_MSGOBJ *)C1FIFOUA1L;
while(C1FIFOSTA1bits.TFNRFNIF ==0);
//Process the received messages
C1FIFOCON1Lbits.UINC = 1;                   // Update the FIFO message pointer.
while(1);
}
```

10.0 FIFO BEHAVIOR

This section explains the FIFO behavior when TEF and TXQ are enabled. FIFO 1 is configured as a TX FIFO and FIFO 2 as an RX FIFO. The remaining FIFOs are not configured.

Note 1: The start addresses are calculated based on the number of objects in the FIFO and the PLSIZEx bits.

2: The start addresses of the FIFOs given in Table 10-1 are calculated when TEF starts at 0x1400.

Table 10-1: Example FIFO Configuration

FIFO	Objects in FIFO	Payload per Object	Timestamp	Bytes in Object	Bytes in FIFO	Start Address
TEF	12	N/A	Yes	12	144	0x1400
TXQ	8	32	N/A	40	320	0x1490
FIFO 1	5	64	N/A	72	360	0x15D0
FIFO 2	16	64	Yes	76	1216	0x1738
FIFO 3	N/A	—	—	—	—	0x1BF8

10.1 FIFO Status Flags

FIFO 1 through FIFO 31 can be configured as transmit or receive FIFOs. The same status flags in CxFIFOSTAx are used for transmit and receive FIFOs. The status flags behave differently based on the selected configuration.

10.1.1 TX FIFO STATUS FLAGS

There are three transmit status flags:

- TFEIF (**TFE**RFF**IF**): Transmit FIFO Empty Interrupt Flag; set when the FIFO is empty.
- TFHIF (**TFH**RFH**IF**): Transmit FIFO Half Empty Interrupt Flag; set when FIFO is less than half full.
- TFNIF (**TFN**RFN**IF**): Transmit FIFO Not Full Interrupt Flag; set when FIFO is not full.

The status flags of a transmit FIFO are set when there is space to load a new message object into the FIFO. Before the first message object is loaded (after the FIFO is reset), all status flags are set. When the FIFO is fully loaded, all flags are cleared.

10.1.2 RX FIFO STATUS FLAGS

There are three receive status flags:

- RFFIF (TFE**RFFIF**): Receive FIFO Full Interrupt Flag; set when the FIFO is full.
- RFHIF (TFH**RFHIF**): Receive FIFO Half Full Interrupt Flag; set when the FIFO is at least half full.
- RFNIF (TFN**RFNIF**): Receive FIFO Not Empty Interrupt Flag; set when there is at least one message in the FIFO.

The status flags of the receive FIFO are set when there are received messages in the FIFO. Before the first message is received (after the FIFO is reset), all status flags are cleared. When the FIFO is full, all flags are set.

10.1.3 TXQ STATUS FLAGS

There are two TXQ status flags:

- TXQEIF: TXQ Empty Interrupt Flag; set when the TXQ is empty.
- TXQNIF: TXQ Not Full Interrupt Flag; set when TXQ is not full.

The status flags of the TXQ are set when there is space to load a new message object into the TXQ. Before the first message object is loaded (after the TXQ is reset), all status flags are set. When the TXQ is fully loaded, all flags are cleared.

10.1.4 TEF STATUS FLAGS

There are four TEF status flags:

- TEFFIF: TEF Full Interrupt Flag; set when the TEF is full.
- TEFHIF: TEF Half Full Interrupt Flag; set when the TEF is at least half full.
- TEFNEIF: TEF Not Empty Interrupt Flag; set when there is at least one message in the TEF.
- TEFOVIF: TEF Overrun Interrupt Flag; set when an overflow has occurred.

The status flags of the TEF are set when there are transmitted messages in the FIFO. Before the first message is stored (after the TEF is reset), all status flags are cleared. When the TEF is full, all flags are set.

10.2 Transmit FIFO Behavior

FIFO 1 is configured as a TX FIFO. CxFIFOCON1L and CxFIFOCON1H are used to control the FIFO. CxFIFOSTA1 contains the status flags and the FIFO Index bits (FIFOCI[4:0]). CxFIFOUA1L and CxFIFOUA1H contain the user address of the next transmit message object to be loaded.

Figure 10-1 through Figure 10-6 illustrate how the status flags, user address and FIFO index are updated for FIFO 1.

Figure 10-1 shows the status of FIFO 1 after Reset. Message objects, MO0 to MO4, are empty. All status flags are set. The user address and the FIFO index point to MO0.

Figure 10-1: FIFO 1 – Initial State

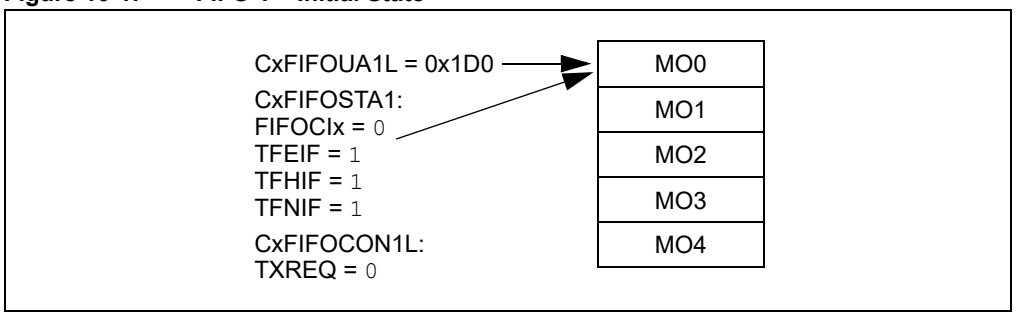


Figure 10-2 illustrates the status of FIFO 1 after the first message (MSG0) is loaded. MO0 now contains MSG0. The user application sets the UINC bit (CxFIFOCON1L[8]), which causes the FIFO head to advance. The user address now points to MO1. TFEIF is cleared since the FIFO is no longer empty. The user application now sets TXREQ to request the transmission of MSG0.

Figure 10-2: FIFO 1 – First Message Loaded

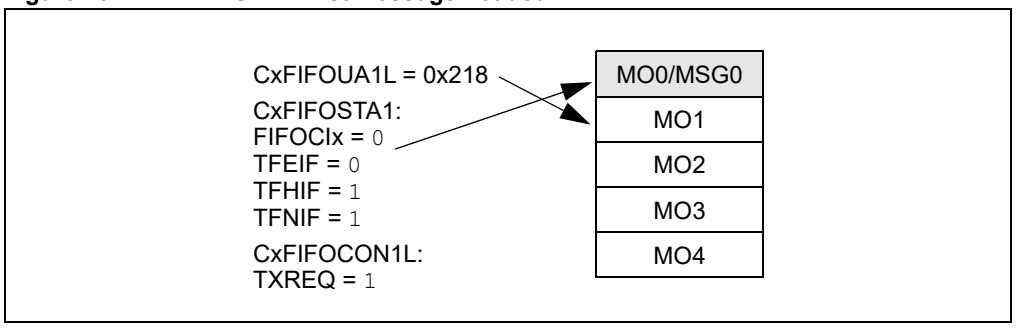


Figure 10-3 illustrates the status of FIFO 1 after MSG0 is transmitted. The FIFO is empty again. TFEIF is set and TXREQ is cleared. FIFOCIx bits now point to MO1 with user address 0x218.

Figure 10-3: FIFO 1 – First Message Transmitted

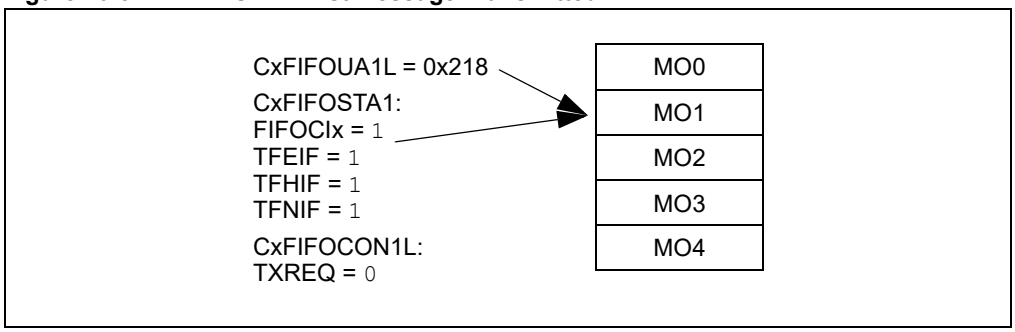


Figure 10-4 illustrates the status of FIFO 1 after three more messages are loaded: MSG1-MSG3. The user address now points to MO4. TFHIF is cleared because the FIFO is now less than half empty.

Figure 10-4: FIFO 1 – Three More Messages Loaded

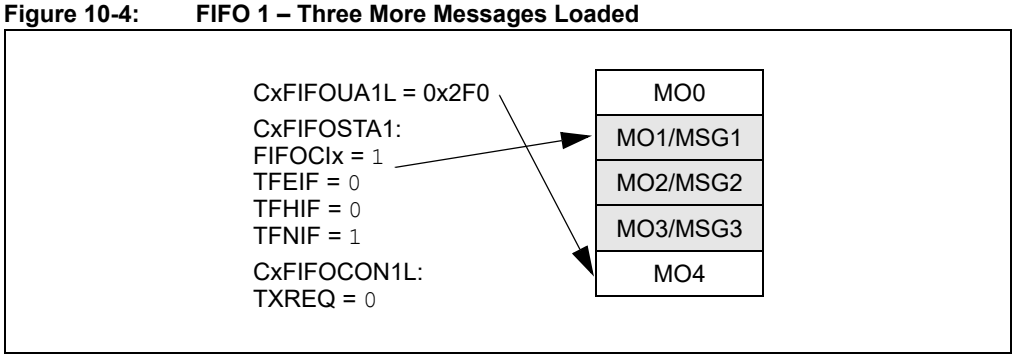


Figure 10-5 illustrates the status of FIFO 1 after two more messages are loaded: MSG4 and MSG5. CxFIFOUA1L now points to MO1. All status flags are now cleared because the FIFO is full. The user address and the FIFO index now point to MO1. The user application now sets TXREQ to request the transmission of MSG1-MSG5.

Figure 10-5: FIFO 1 – FIFO Fully Loaded

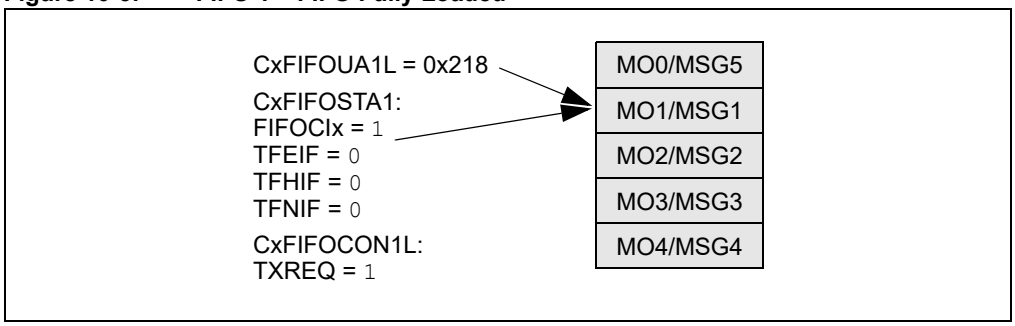
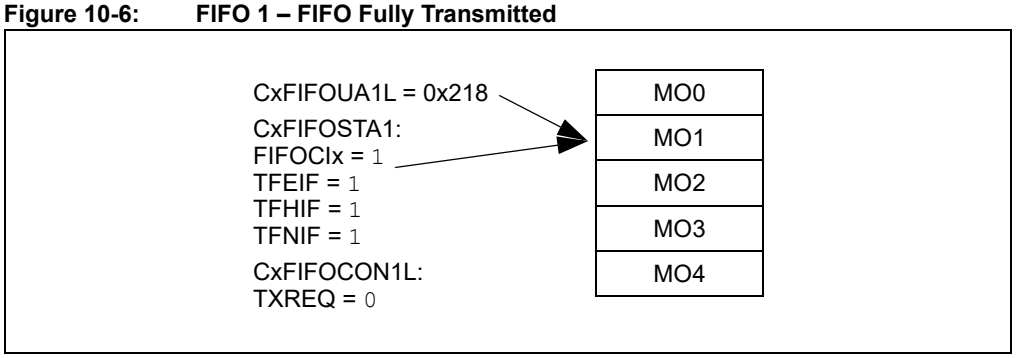


Figure 10-6 illustrates the status of FIFO 1 after MSG1-MSG5 are transmitted. The FIFO is empty again. All status flags are set and TXREQ is cleared. The user address and the FIFO index point to MO1 again.

Figure 10-6: FIFO 1 – FIFO Fully Transmitted



10.3 Receive FIFO Behavior

FIFO 2 is configured as an RX FIFO. CxFIFOCON2L and CxFIFOCON2H are used to control the FIFO. CxFIFOSTA2 contains the status flags and the FIFO index (FIFOCIx). CxFIFOUA2L and CxFIFOUA2H contain the user address of the next message object to read.

Figure 10-7 through Figure 10-14 illustrate how the status flags, user address and FIFO index are updated.

Figure 10-7 shows the status of FIFO 2 after the Reset. Message objects, MO0 to MO15, are empty. All status flags are cleared. The user address and the FIFO index point to MO0.

Figure 10-7: FIFO 2 – Initial State

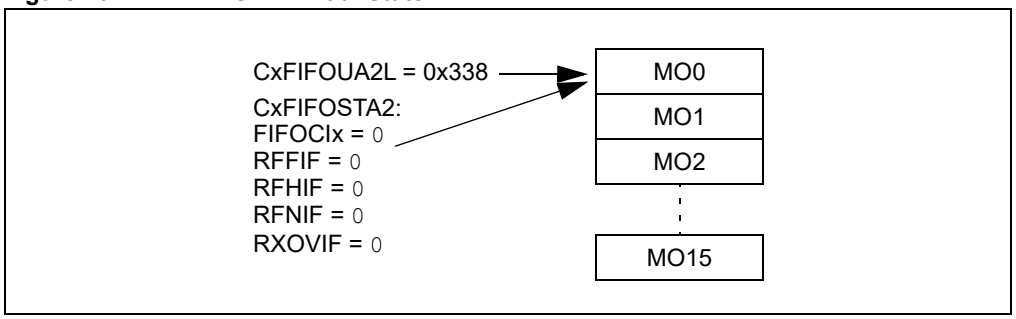


Figure 10-8 illustrates the status of FIFO 2 after the first message (MSG0) is received. MO0 now contains MSG0. The FIFO index now points to MO1. RFNIF is set since the FIFO is not empty anymore.

Figure 10-8: FIFO 2 – First Message Received

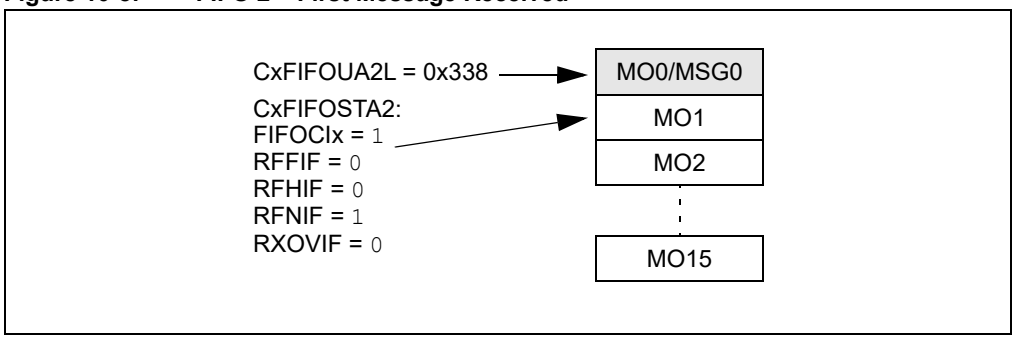


Figure 10-9 illustrates the status of FIFO 2 after MSG0 is read. The user application reads the message from RAM and sets the UINC bit (CxFIFOCON2L[8]). The user address increments and points to MO1. The FIFO index is unchanged. The FIFO is empty again. All flags are cleared.

Figure 10-9: FIFO 2 – First Message Read

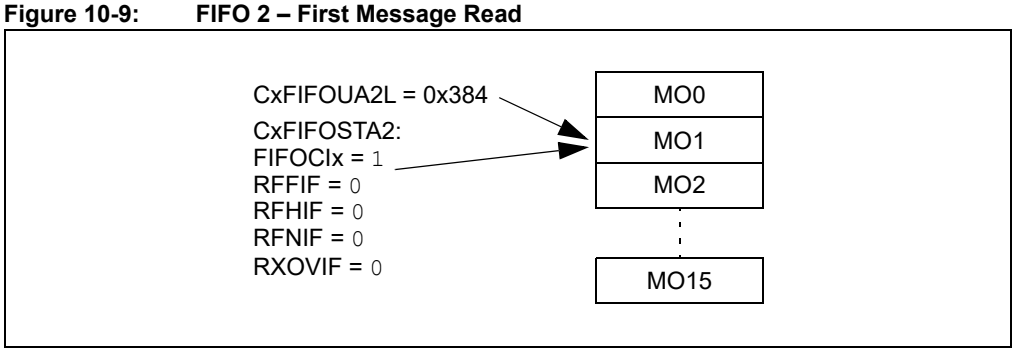


Figure 10-10 illustrates the status of FIFO 2 after eight more messages are received: MSG1-MSG8. The user address still points to MO1. RFNIF and RFHIF are set because the FIFO is now half full. The FIFO index points to MO9.

Figure 10-10: FIFO 2 – Half Full

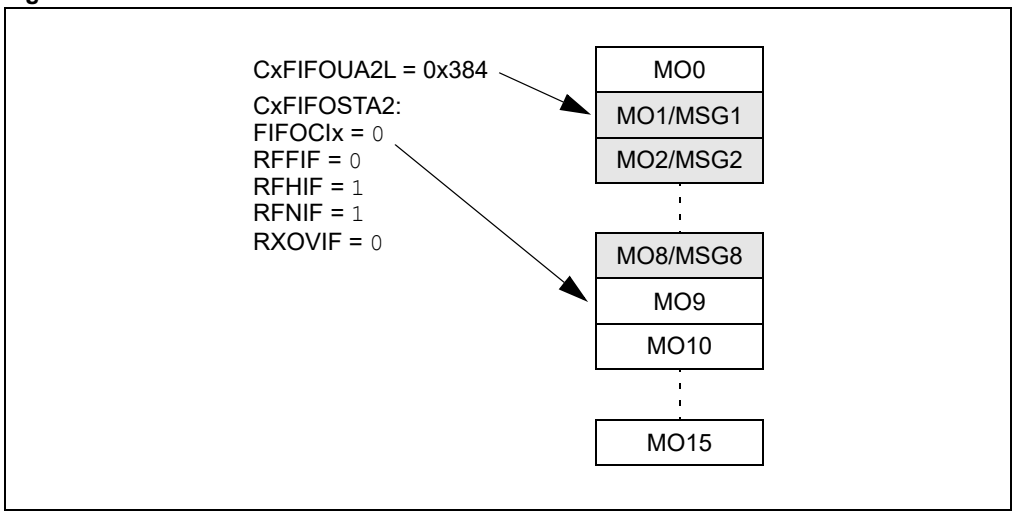


Figure 10-11 illustrates the status of FIFO 2 after ten more messages are received: MSG5-MSG15. The user address still points to MO1. The FIFO index points to MO0. RFNIF and RFHIF are set.

Figure 10-11: FIFO 2 – FIFO Almost Full

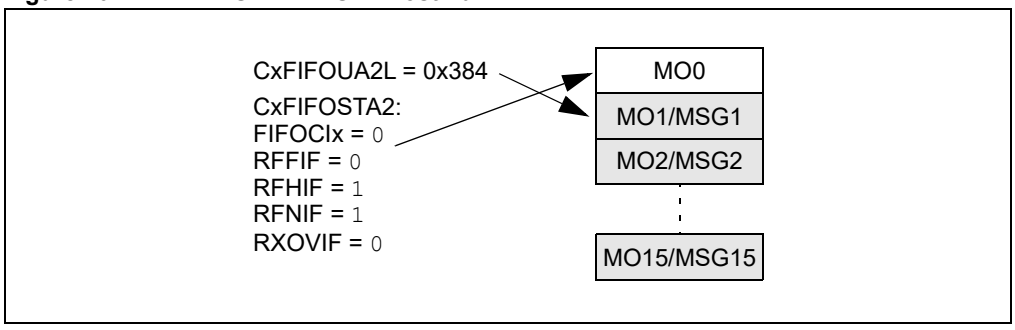


Figure 10-12 illustrates the status of FIFO 2 after one more message is received: MSG16. All status flags are set because the FIFO is full. The user address and the FIFO index point to MO1.

Figure 10-12: FIFO 2 – FIFO Full

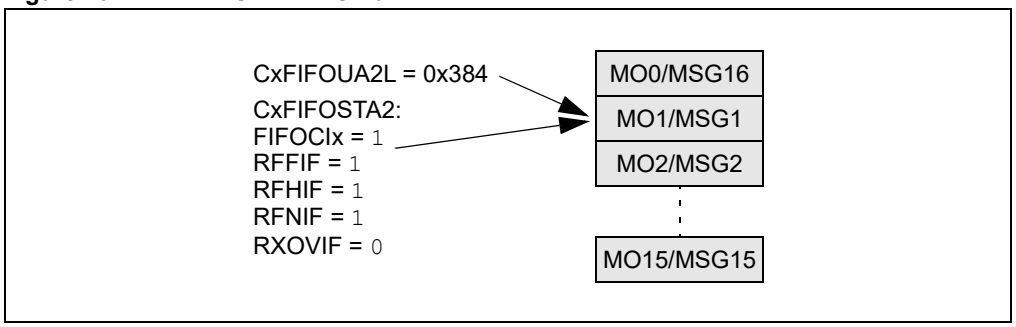


Figure 10-13 illustrates the status of FIFO 2 after one more message is received. Since FIFO 2 is already full, an overflow occurs. The message is discarded and RXOVIF is set. The user address and FIFO index has not changed.

Figure 10-13: FIFO 2 – FIFO Overflow

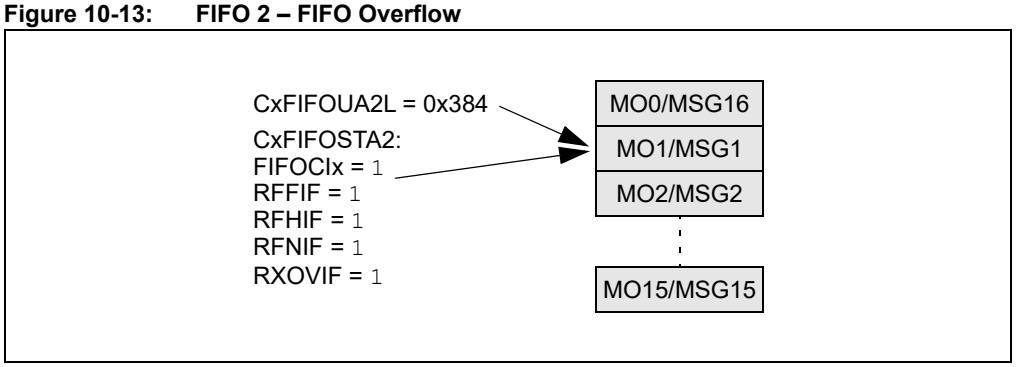
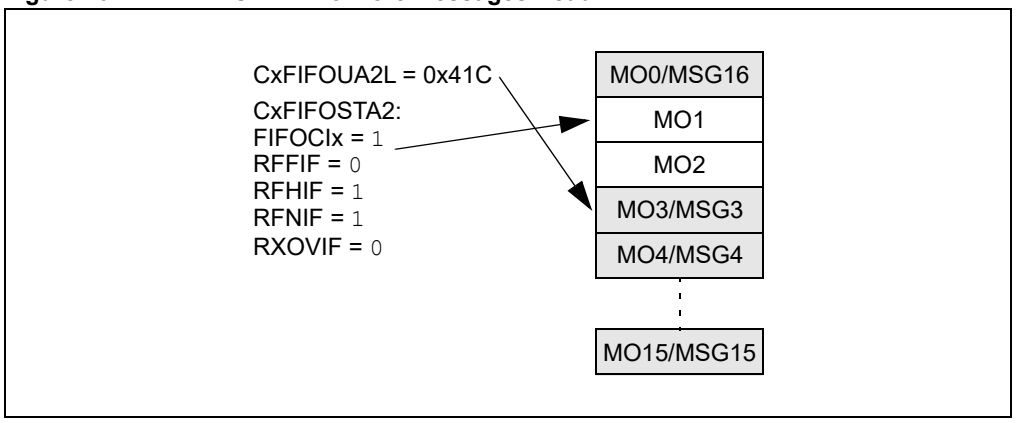


Figure 10-14 illustrates the status of FIFO 2 after the application cleared RXOVIF and read two more messages. RFFIF is clear because the FIFO is not full anymore. The user address points to MO3. The FIFO index has not changed.

Figure 10-14: FIFO 2 – Two More Messages Read



10.4 Transmit Queue Behavior

CxTXQCONL and CxTXQCONH are used to control the TXQ. CxTXQSTA contains the status flags and the TXQ index (TXQCIx). CxTXQUAL and CxTXQUAH contain the user address of the next transmit message object to be loaded.

The TXQCI[4:0] bits are used by the CAN FD Protocol Module to calculate the next message to transmit. TXQCIx bits are not incremented linearly. They are recalculated every time a message gets transmitted or TXREQ gets set.

Figure 10-15 through Figure 10-20 illustrate how the status flags and user address are updated. There is no need for the user application to use TXQCIx; therefore, it is not shown in the figures.

Figure 10-15 shows the status of the TXQ after Reset. Message objects, MO0 to MO7, are empty. All status flags are set. The user address points to MO0.

Figure 10-15: TXQ – Initial State

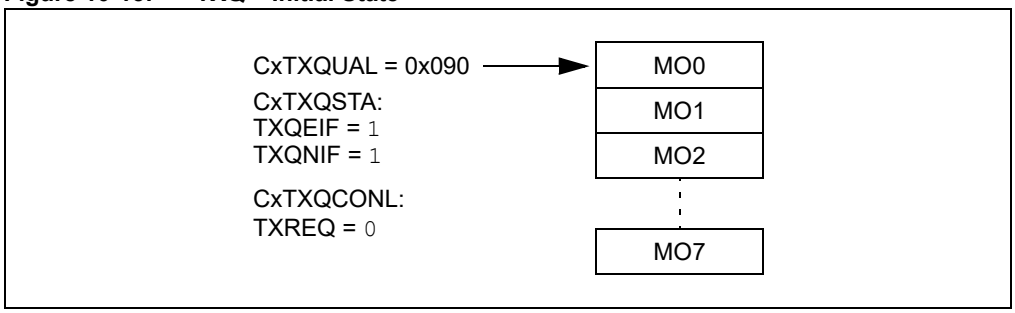


Figure 10-16 illustrates the status of the TXQ after the first message (MSG0) is loaded. MO0 now contains MSG0. The user application sets the UINC bit, which causes the FIFO head to advance. The user address now points to MO1. TXQEIF is cleared, since the queue is not empty anymore. The user application now sets TXREQ to request the transmission of MSG0.

Figure 10-16: TXQ – First Message Loaded

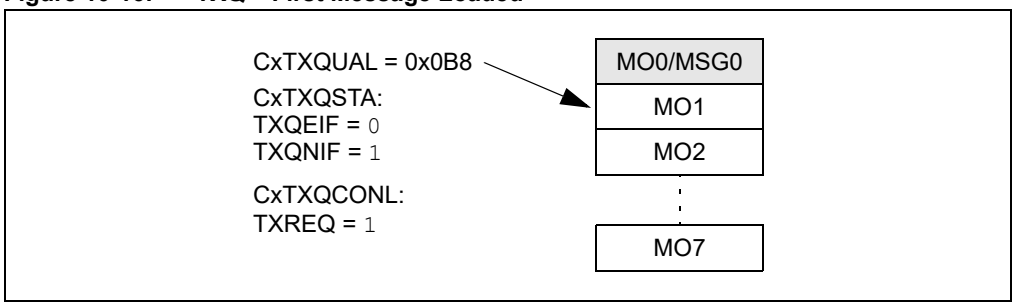


Figure 10-17 illustrates the status of the TXQ after MSG0 is transmitted. The TXQ is empty again. TXQEIF is set and TXREQ is cleared. The user address still points to MO1 because UINC is not set.

Figure 10-17: TXQ – First Message Transmitted

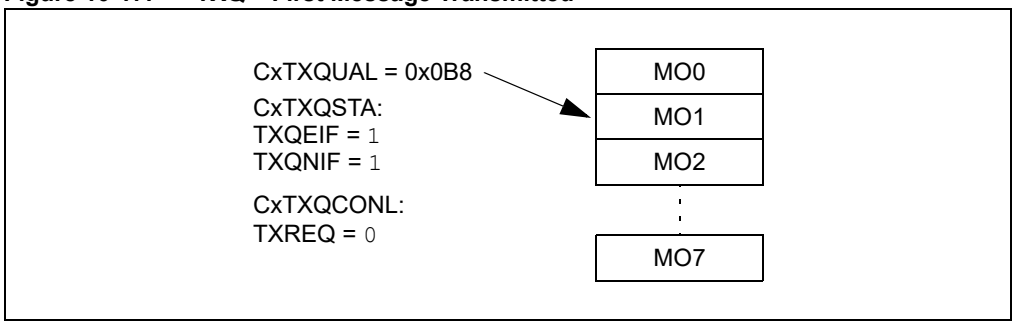


Figure 10-18 illustrates the status of the TXQ after MSG1 is loaded and UINC is set. The user address now points to the next free message object: MO0.

Figure 10-18: TXQ – Next Message Loaded

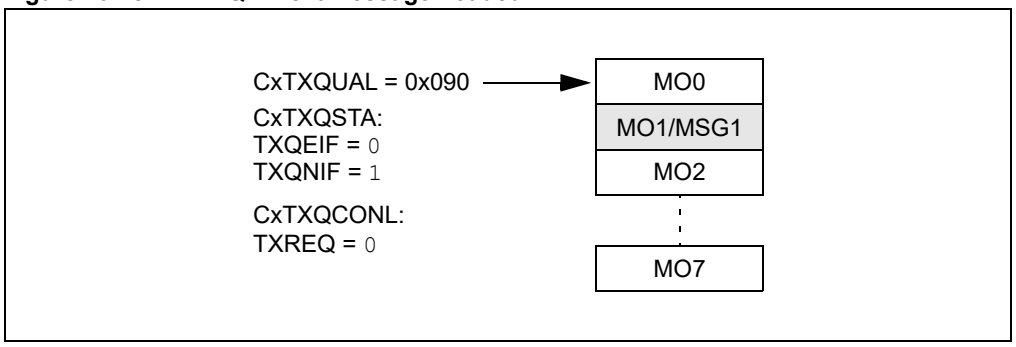


Figure 10-19 illustrates the status of the TXQ after six more messages are loaded: MSG2-MSG7. The user address now points to the last free message object: MO7.

Figure 10-19: TXQ – Next Six Messages Loaded

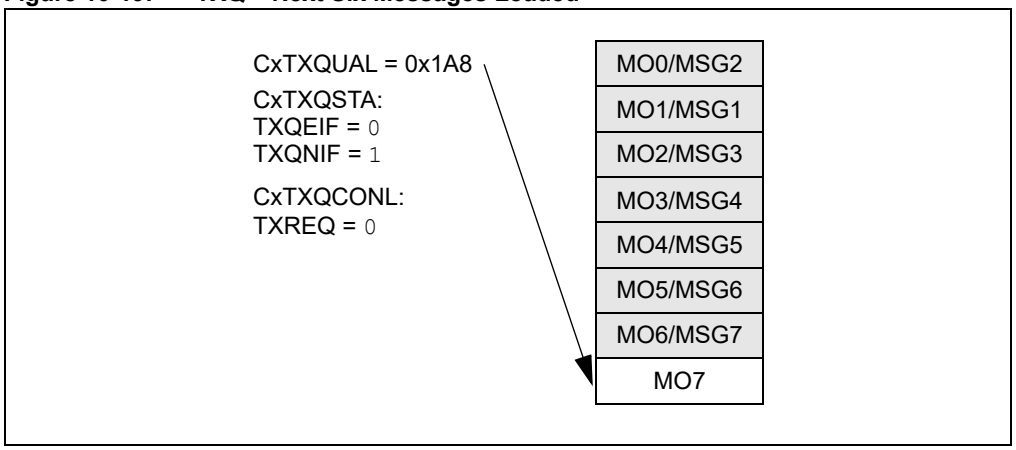
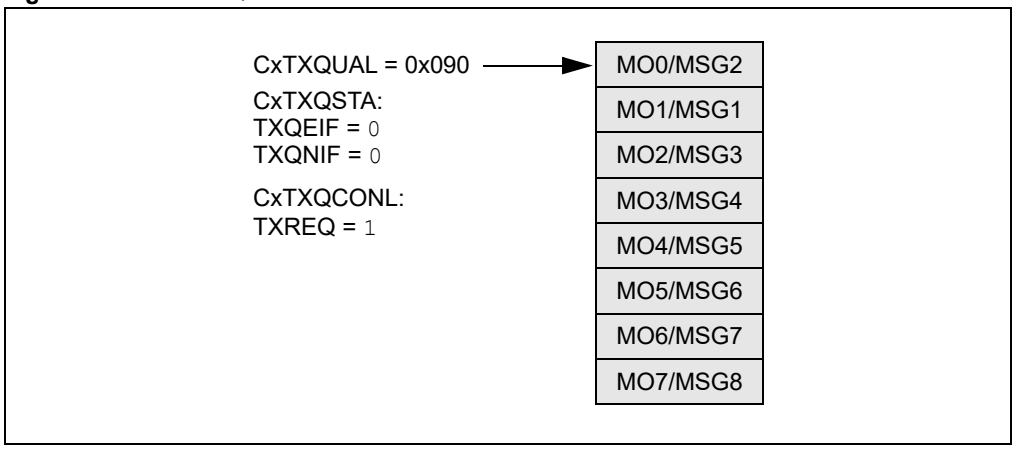


Figure 10-20 illustrates the status of the TXQ after MSG8 is loaded and UINC is set. The TXQ is now full, all flags are cleared. The user address now points to MO0. The user application now sets TXREQ. The messages will be transmitted based on the priority of their IDs.

Figure 10-20: TXQ – Full



10.5 Transmit Event FIFO Behavior

CxTEFCONL and CxTEFCONH are used to control the TEF. CxTEFSTA contains the status flags. CxTEFUAL and CxTEFUAH contain the user address of the next message object to read.

The actual RAM address is calculated using Equation 7-1.

Figure 10-21 through Figure 10-28 illustrate how the status flags and user address are updated. The TEF stores transmitted messages; therefore, the flags behave similarly to an RX FIFO.

Figure 10-21 shows the status of the TEF after Reset. Message objects, MO0 to MO11, are empty. All status flags are cleared. The user address points to MO0.

Figure 10-21: TEF – Initial State

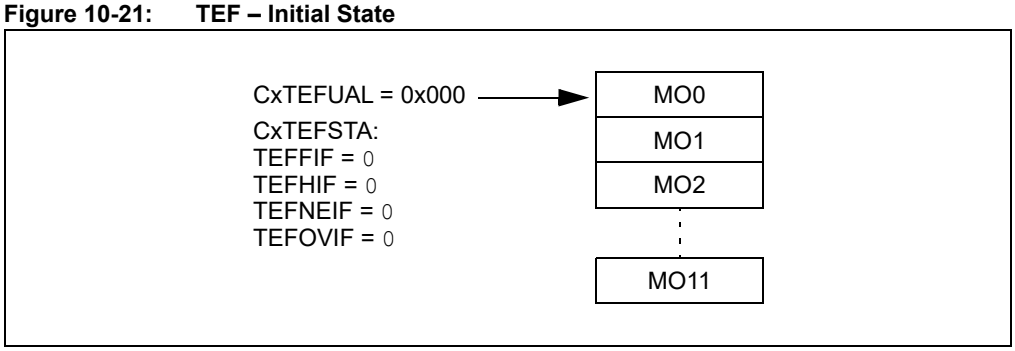


Figure 10-22 shows the status of the TEF after the first transmit message is stored. MO0 contains ID0, the ID of MSG0. TEFNEIF is set since the TEF is not empty. The user address points to MO0.

Figure 10-22: TEF – First Transmit Message is Stored

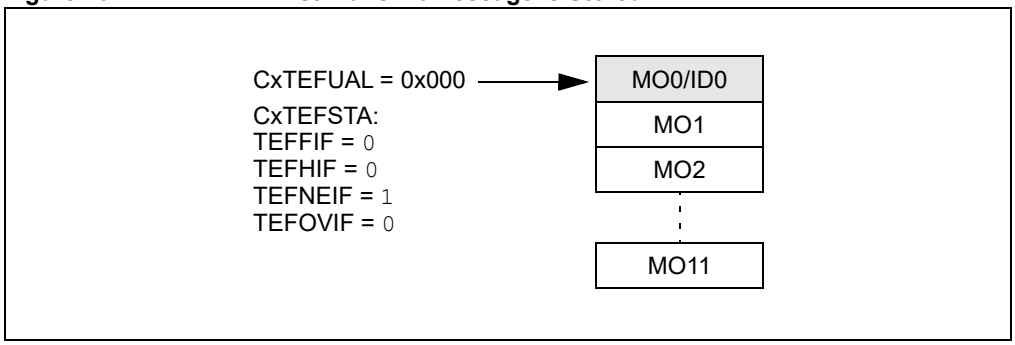


Figure 10-23 illustrates the status of the TEF after ID0 is read. The user application reads the ID from RAM and sets the UINC bit (CxTEFCONL[8]). The user address increments and points to MO1. The TEF is empty again. All flags are cleared.

Figure 10-23: TEF – First ID Read

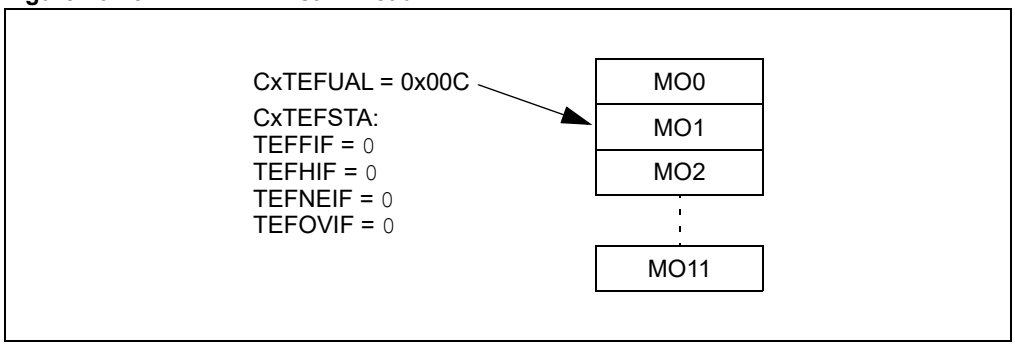


Figure 10-24 illustrates the status of the TEF after six more messages are transmitted: MSG1-MSG6. The user address points to MO1. TEFNEIF and TEFHIF are set because the TEF is now half full.

Figure 10-24: TEF – Half Full

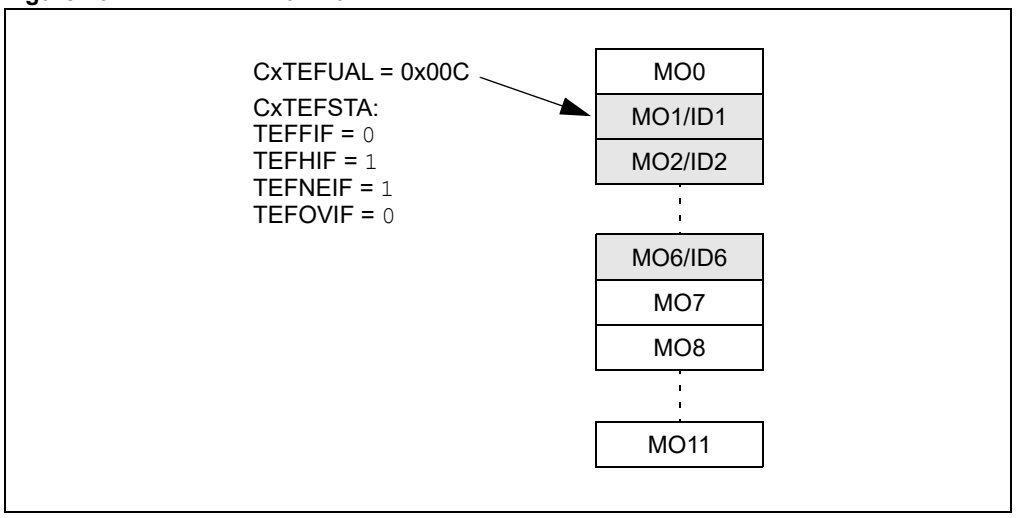


Figure 10-25 illustrates the status of the TEF after five more messages are transmitted: MSG7-MSG11. The user address still points to MO1. TEFNEIF and TEFHIF are set.

Figure 10-25: TEF – Almost Full

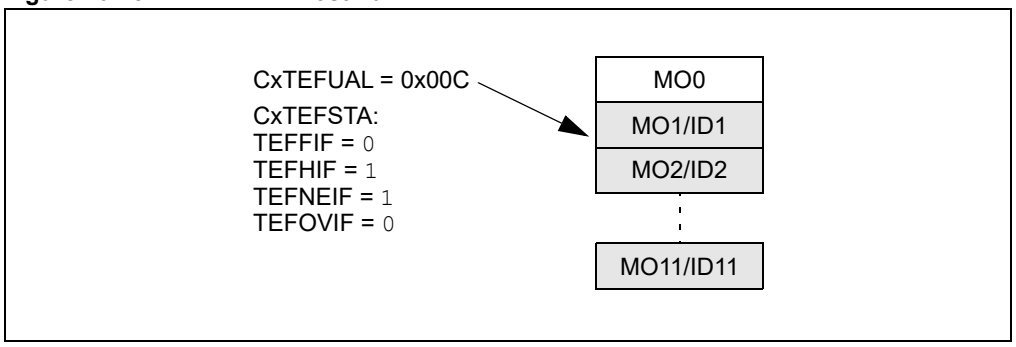


Figure 10-26 illustrates the status of the TEF after one more message is transmitted: MSG12. All status flags are set because the TEF is full. The user address points to MO1.

Figure 10-26: TEF – Full

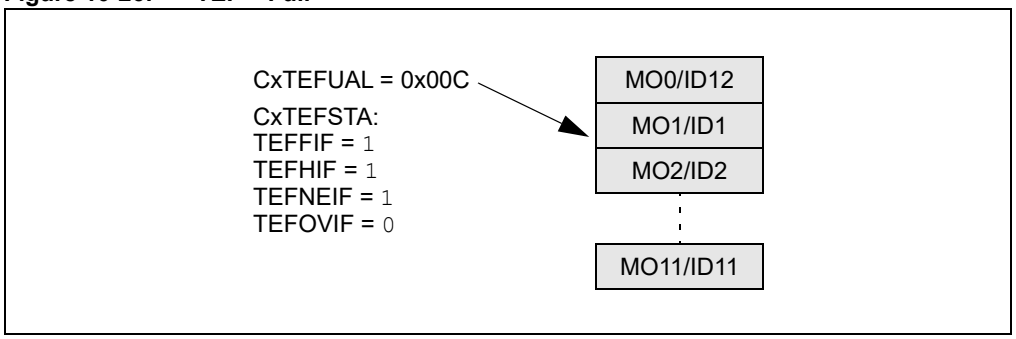


Figure 10-27 illustrates the status of the TEF after one more message is transmitted. Since the TEF is already full, an overflow occurs. The ID is discarded and TEFOVIF is set. The user address remains unchanged.

Figure 10-27: TEF – Overflow

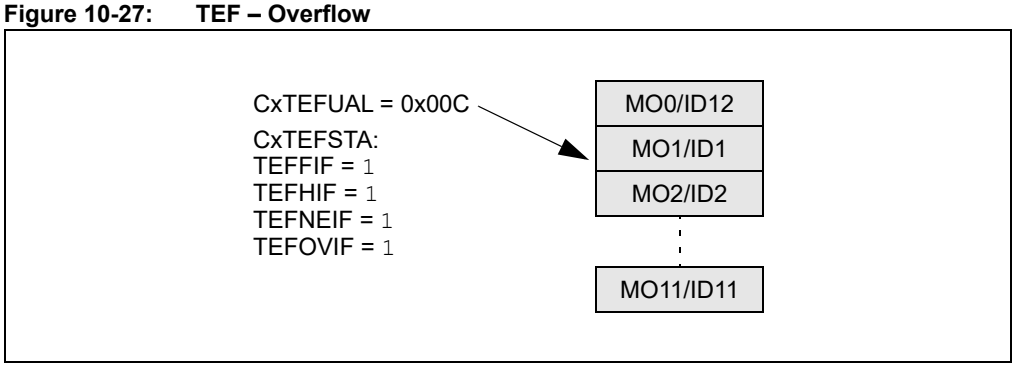
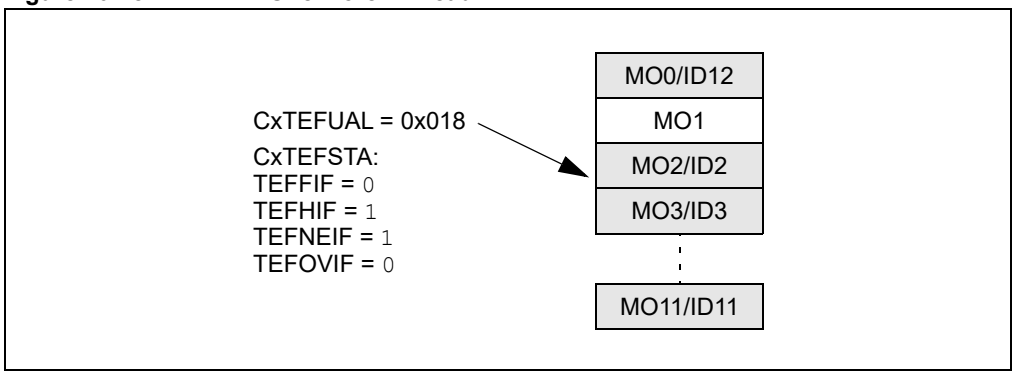


Figure 10-28 illustrates the status of the TEF after the application cleared TEFOVIF and read one more message. TEFFIF is clear because the TEF is not full anymore. The user address points to MO2.

Figure 10-28: TEF – One More ID Read



11.0 TIMESTAMPING

The CAN FD Protocol Module contains a Time Base Counter (TBC). The TBC is a 32-bit free-running counter that increments on multiples of FCAN and rolls over to zero when:

- TBCPRE[9:0] bits (CxTSCONL[9:0]) are used to configure the prescaler for the TBC
- Setting TBCEN (CxTSCONH[0]) enables the TBC
- Clearing TBCEN disables, stops and resets the TBC
- The TBC has to be disabled before writing to CxTBCL/H by clearing TBCEN
- TEFTSEN (CxTEFCONL[5]) has to be set to timestamp messages in the TEF
- RXTSEN (CxFIFOCONxL[5]) has to be set to timestamp messages in the individual RX FIFO
- The application can read CxTBCL/H at any time. Similar to any multibyte counter, the application has to consider that the counter increments and might roll over while reading different bytes of the counter.

All timestamps are 32 bits, allowing timestamps to be used for system time synchronization with high resolution.

A rollover of the TBC will generate an interrupt if TBCIE is set.

Messages can be timestamped either at the beginning of a frame or at the end, depending on the TSEOF bit (CxTSCONH[1]). When TSEOF = 0, TSRES (CxTSCONH[2]) specifies if FD frames are timestamped at SOF or the “reserved bit”. Table 11-1 specifies the reference points when the timestamping occurs. At the reference point, the value of the TBC (CxTBCL/H) is captured and stored into the message object:

- Receive Message Object: The TBC value is stored in the RXMSGTSx bits (see Table 9-1)
- TEF Object: The TBC value is stored in the TXMSGTSx bits (see Table 7-1)

Table 11-1: Reference Point

Frame	CAN 2.0	CAN FD
Start of TX	Sample point of SOF	Sample point of SOF or the bit after FDF
Start of RX	Sample point of SOF	Sample point of SOF or the bit after FDF
Valid TX	No error till end of EOF	No error till end of EOF
Valid RX	No error till the last, but one bit of EOF	No error till the last, but one bit of EOF

12.0 INTERRUPTS

Interrupts can be classified into multiple layers. Lower layer interrupts propagate to higher layers by multiplexing them into single interrupts. Figure 12-1 illustrates the layers of interrupts.

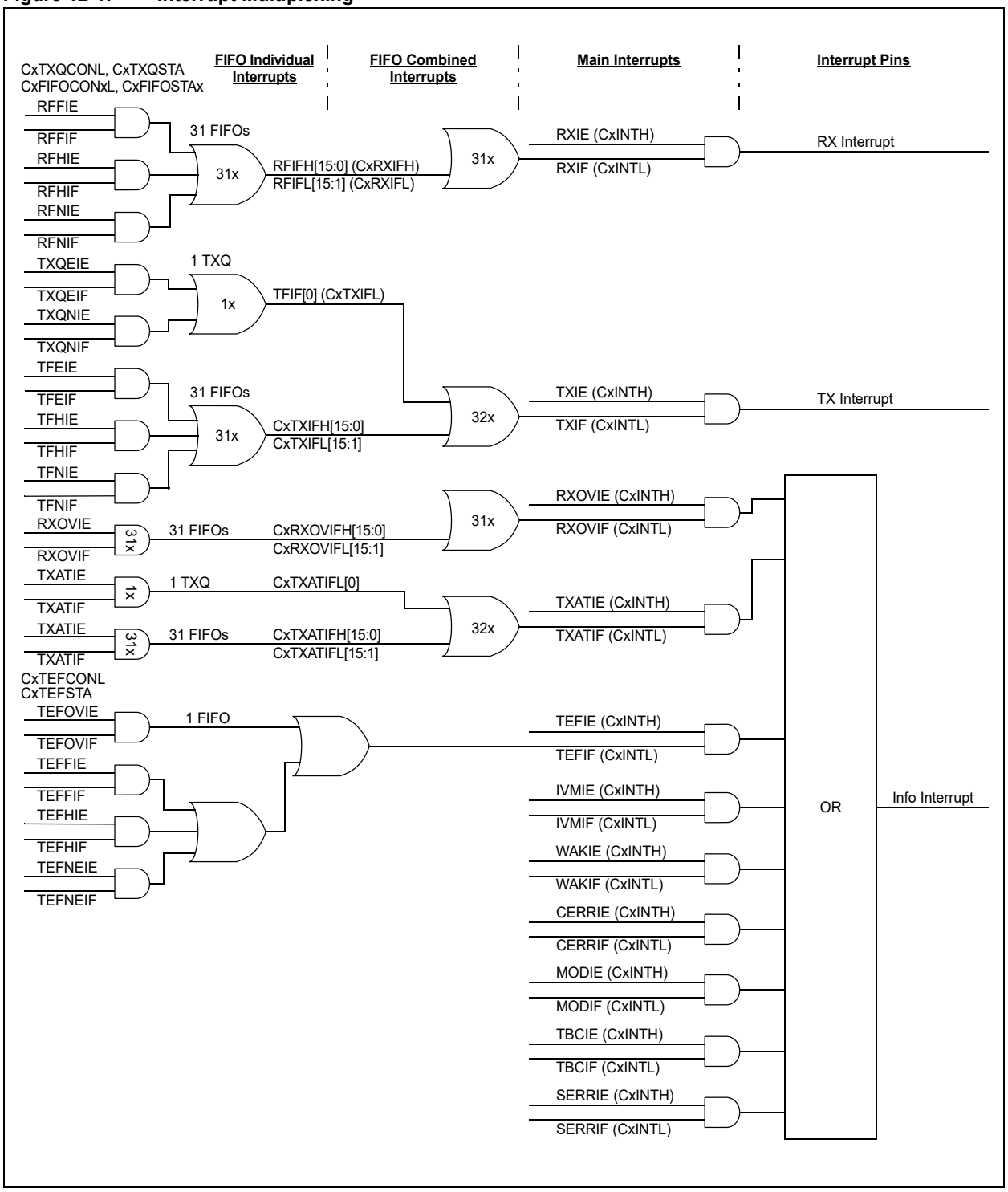
- FIFO Individual Interrupts
- FIFO Combined Interrupts
- Main Interrupts

These interrupts are then funneled into three separate module interrupts:

- Receive Interrupt
- Transmit Interrupt
- Information Interrupt

All module interrupts are persistent, meaning the condition that caused the interrupt must be cleared within the module for the interrupt request to be removed.

Figure 12-1: Interrupt Multiplexing



12.1 FIFO Individual Interrupts

CxFIFOCONxL contains the interrupt enable flags and CxFIFOSTAx contains the interrupt flags for the FIFOs. There is a separate register for each FIFO.

12.1.1 TRANSMIT QUEUE INTERRUPTS

CxTXQCONL contains the interrupt enable flags and CxTXQSTA contains the interrupt flags for the TXQ.

The TXQ interrupt occurs when there is a change in the status of the TXQ. There are two interrupt sources:

- TXQ Not Full Interrupt Flag (TXQNIF)
- TXQ Empty Interrupt Flag (TXQEIF)

Both interrupts can be enabled individually. The interrupts cannot be cleared by the application; they will be cleared when the condition of the FIFO terminates.

Both interrupt sources are OR'd together and reflected in the TFIF0 flag (CxTXIFL[0]).

12.1.2 RECEIVE FIFO INTERRUPTS – RFIF

The receive FIFO interrupts occur when there is a change in the status of the receive FIFO. There are three interrupt sources:

- Receive FIFO Full Interrupt Flag (RFFIF)
- Receive FIFO Half Full Interrupt Flag (RFHIF)
- Receive FIFO Not Empty Interrupt Flag (RFNIF)

All three interrupts can be enabled individually. The interrupts cannot be cleared by the application; they will be cleared when the condition of the FIFO terminates.

The three interrupt sources are OR'd together and reflected in the RFIF[31:16] (CxRXIFH[15:0]) and RFIF[15:1] (CxRXIFL[15:1]) flags.

12.1.3 TRANSMIT FIFO INTERRUPTS – TFIF

The transmit FIFO interrupts occur when there is a change in the status of the transmit FIFO. There are three interrupt sources:

- Transmit FIFO Not Full Interrupt Flag (TFNIF)
- Transmit FIFO Half Empty Interrupt Flag (TFHIF)
- Transmit FIFO Empty Interrupt Flag (TFEIF)

All three interrupts can be enabled individually. The interrupts cannot be cleared by the application; they will be cleared when the condition of the FIFO terminates.

The three interrupt sources are OR'd together and reflected in the CxTXIFL[15:1] and CxTXIFH[15:0] flags.

12.1.4 RECEIVE FIFO OVERRUN INTERRUPT – RXOVIF

When a message is successfully received, but the FIFO is full, the RXOVIF of the individual FIFO is set. The flag must be cleared by the application.

12.1.5 TRANSMIT FIFO ATTEMPT INTERRUPT – TXATIF

When the retransmission of a message fails due to an error, and all retransmission attempts are exhausted, the TXATIF flag is set. The flag must be cleared by the application.

12.1.6 TRANSMIT EVENT FIFO INTERRUPTS – TEFIF

The TEF interrupts occur when there is a change in the status of the TEF. There are four interrupt sources:

- TEF Full Interrupt Flag (TEFFIF)
- TEF Half Full Interrupt Flag (TEFHIF)
- TEF Not Empty Interrupt Flag (TEFNEIF)
- TEF Overrun Interrupt Flag (TEFOVIF)

The TEF interrupts work similarly to the receive FIFO interrupts. All four interrupts can be enabled individually.

TEFFIF, TEFHIF and TEFNEIF cannot be cleared by the application; they will be cleared when the status of the FIFO terminates.

The TEFOVIF must be cleared by the application.

The four interrupt sources are OR'd together and reflected in the TEFIF flag (CxINTL[4]).

12.2 FIFO Combined Interrupts

The following interrupts are individual FIFO interrupts:

- FIFOs/TXQ: RFIFx, TFIFx, RFOVIFx and TFATIFx

They are combined into single Interrupt Status registers:

- CxRXIFL/H, CxTXIFL/H, CxRXOVIFL/H and CxTXATIFL/H

The bits in the status registers are mapped to the FIFOs as follows: Bit 0 to TXQ, Bit 1 to FIFO 1, Bit 2 to FIFO 2, up to Bit 31 to FIFO 31. Since Bit 0 corresponds to the TXQ, Bit 0 of CxRXIFL and CxRXOVIFL is reserved. Hence, by reading one register, the application can check the status of all FIFOs for a particular interrupt (e.g., any RFIFx pending).

The FIFO interrupts are enabled in CxFIFOCONxL.

TXQ interrupts are enabled in CxTXQCONL.

Clearing of the FIFO interrupts is explained in **Section 12.1 "FIFO Individual Interrupts"**.

12.3 Main Interrupts

The CxINTL register contains all the main interrupts. The following interrupts are a logical 'OR' of all combined FIFO interrupts: RXIF, TXIF, RXOVIF and TXATIF. These flags are read-only and must be cleared in preceding hierarchies.

The TEFIF is generated in the TEF. This flag is read-only and must be cleared in preceding hierarchies.

All interrupts in CxINTL/H can be enabled individually.

12.3.1 INVALID MESSAGE INTERRUPT – IVMIF

If a CAN bus error or DLC mismatch is detected during the last message transmitted or received, the IVMIF bit will be set. The CxBDIAG1H register sets a flag for each error. The flag must be cleared by the application.

The following CAN bus errors will trigger the interrupt in case an error frame is transmitted: CRC, stuff bit, form, bit or ACK.

The flag will not be set if the ESI of a received message is set.

12.3.2 WAKE-UP INTERRUPT – WAKIF

This bit is set if bus activity has been detected while the module is in Sleep mode. The flag must be cleared by the application.

12.3.3 CAN BUS ERROR INTERRUPT – CERRIF

The CxTRECL/H registers will count the errors during transmit and receive according to the ISO11898-1:2015. The CERRIF flag will be set based on the error counter values. The flag must be cleared by the application.

CERRIF will be set each time a threshold in the TEC/REC counter is crossed by the following conditions:

- TEC or REC exceeds the error warning state threshold
- The transmitter or receiver transitions to the error passive state
- The transmitter transitions to the bus off state
- The transmitter or receiver transitions from the error passive to error active state
- The module transitions from the bus off to error active state after the bus off recovery sequence

When the user clears CERRIF, it will remain clear until a new counter crossing occurs.

12.3.4 CAN MODE CHANGE INTERRUPT – MODIF

When the OPMOD[2:0] bits change, the MODIF flag will be set. The flag must be cleared by the application.

12.3.5 CAN TIMER INTERRUPT – TBCIF

When the Time Base Counter rolls over, TBCIF will be set. The flag must be cleared by the application.

12.3.6 SYSTEM ERROR INTERRUPT – SERRIF

- Bus Bandwidth Error:

  Bandwidth errors can happen during receive and transmit.

  Receive Message Assembly Buffer (RX MAB) overflow occurs when the module is unable to write a received CAN message to RAM before the next message arrives.

  Transmit Message Assembly Buffer (TX MAB) underflow occurs when the module cannot feed the TX MAB fast enough to provide consistent data to the Bit Stream Processor.

  The SERRIF flag will be set and the ICODE[6:0] bits (CxVECL[6:0]) will be set to `100 0101`.

- Handling of RX MAB Overflow Errors:

  RX MAB overflows are not acceptable for some applications. To prevent overflows, frame filtering and data saving starts as early as possible; the latest at the beginning of the CRC field of the received message. Updating the FIFO status has to wait until the beginning of the 7th bit of the EOF field, since the received frame is only valid at this point. The complete message has to be saved and the FIFO has to be updated until the end of the arbitration field of the next message.

  In case of an RX MAB overflow, the new message that caused the overflow will be discarded. The module continues to store the message that is completely received and filtered. Afterwards, the module will be able to receive new messages on the bus. The application will be notified using the SERRIF bit.

  The SERRIF bit (CxINTL[12]) will be cleared by writing a zero to the bit. This will also clear the SERRIF condition from the ICODEx bits.

- Handling of TX MAB Underflow Errors:

  ISO11898-1:2015 requires MAC data consistency: a transmitted message must contain consistent data. If data errors occur due to ECC errors, or TX MAB underflow, the transmission will not start. If the transmission is in progress, it will stop and the module will transition to either Restricted Operation or Listen Only mode, which is selectable using the SERRLOM bit (CxCONH[2]).

The module handles these errors by stopping the transmission and transitioning to Restricted Operation or Listen Only mode. The CxTX pin will be forced high. Additionally, all TXREQs will be ignored. The application will be notified using SERRIF. The module will continue to receive messages.

Note:	There are two types of addressing errors and both of them will cause a soft trap error on a dsPIC33C device by setting the CAN bit in the INTCON3 register. The first addressing error occurs when a FIFO is configured with an invalid address. This error most commonly occurs when the FIFO points to an unimplemented address. The second addressing error commonly occurs when the message destination is illegal; for example, attempting to write a received message to a program Flash, which is not directly writable.

12.4 Interrupt Handling

The CAN FD Protocol Module allows the application to handle interrupts efficiently by:

- Implementing a lookup table using the CxVECL/H registers
- Using the status registers and deciding which interrupt to service first

The application can also use a combination of these two methods to handle interrupts.

12.4.1 INTERRUPT LOOKUP TABLE

The ICODEx and FILHITx bits in the CxVECL register enable the application to use a lookup table to implement the Interrupt Service Routine (ISR).

The following bit fields allow the application to make full use of the three interrupt pins:

- TXCODE[6:0] bits: Reflect which object has a transmit interrupt pending
- RXCODE[6:0] bits: Reflect which object has a receive interrupt pending

A separate lookup table can be implemented for transmit and receive interrupts.

If more than one object has a pending interrupt, the interrupt or FIFO with the highest number will show up in RXCODEx, TXCODEx and ICODEx. Once the interrupt with the highest priority is cleared, the next highest priority interrupt will show up in CxVECL/H. RXCODEx, TXCODEx and ICODEx are implemented with combinatorial logic using the interrupt flags as inputs.

12.4.2 INTERRUPT STATUS REGISTERS

The CAN FD Protocol Module contains 31 FIFOs and a TXQ. It would be complex to use the ICODEx bits since the interrupt priorities are determined by the module. Therefore, the following measures are taken to ensure efficient servicing of interrupts:

- CxINTL and CxINTH contain all main interrupt sources. The application can identify the categories of interrupts that are pending and decide the order in which interrupts are to be serviced (e.g., RXIF).
- All categories of interrupts for all FIFOs are combined into individual registers: CxRXIFL/H, CxTXIFL/H, CxRXOVIFL/H and CxTXATIFL/H. The application can identify the RFIFx bits that are pending by reading only one register. The same is true for TFIFx, RXOVIF and TXATIF.
- In the register map, the Interrupt Status registers are arranged in a single block: CxVECL/H, followed by CxINTL/H, CxRXIFL/H, CxTXIFL/H, CxRXOVIFL/H and CxTXATIFL/H. This arrangement allows all status registers to be read with a single read access.

12.5 Interrupt Flags

Table 12-1 summarizes all interrupt flags and lists how interrupts are cleared.

Table 12-1: Interrupt Flags

Flags	Registers	Categories	Cleared by Module(1)	Cleared by Application	Read-Only(2)	Description
RFFIF, RFHIF, RFNIF	CxFIFOSTAx	FIFO	X	—	—	RX FIFO
TFNIF, TFHIF, TFEIF	CxFIFOSTAx	FIFO	X	—	—	TX FIFO
TXQNIF, TXQEIF	CxTXQSTA	TXQ	X	—	—	Transmit Queue
RXOVIF	CxFIFOSTAx	FIFO	—	X	—	RX Overrun
TXATIF	CxFIFOSTAx, CxTXQSTA	FIFO, TXQ	—	X	—	TX Attempt
TEFFIF, TEFHIF, TEFNEIF	CxTEFSTA	FIFO	X	—	—	TEF
TEFOVIF	CxTEFSTA	FIFO	—	X	—	TEF Overrun
RFIF[31:1]	CxRXIFL/H	Combined	—	—	X	All RX FIFOs
TFIF[31:1]	CxTXIFL/H	Combined	—	—	X	All TX FIFOs
RFOVIF[31:1]	CxRXOVIFL/H	Combined	—	—	X	All RX FIFO Overruns
TFATIF[31:0]	CxTXATIFL/H	Combined	—	—	X	All TX FIFO Attempts
RXIF	CxINTL	Main	—	—	X	RX
TXIF	CxINTL	Main	—	—	X	TX
RXOVIF	CxINTL	Main	—	—	X	RX Overrun
TXATIF	CxINTL	Main	—	—	X	TX Attempt
TEFIF	CxINTL	Main	—	—	X	TEF
IVMIF	CxINTL	Main	—	X	—	Invalid Message
WAKIF	CxINTL	Main	—	X	—	Wake-up
CERRIF	CxINTL	Main	—	X	—	CAN Bus Error
MODIF	CxINTL	Main	—	X	—	Mode Change
TBCIF	CxINTL	Main	—	X	—	Time Base Counter
SERRIF	CxINTL	Main	—	X	—	System Error

Note 1: The flags will be cleared when the condition of the FIFO terminates, initiated by the UINC bit (CxFIFOCONxL[8].

2: The flags need to be cleared in the preceding hierarchies.

13.0 ERROR HANDLING

Every CAN controller checks the messages on the bus for the following errors: bit, stuff, CRC, form and ACK errors. Whenever the controller detects an error, an error frame is transmitted that deletes the message on the bus. Error frames are always signaled using the Nominal Bit Rate.

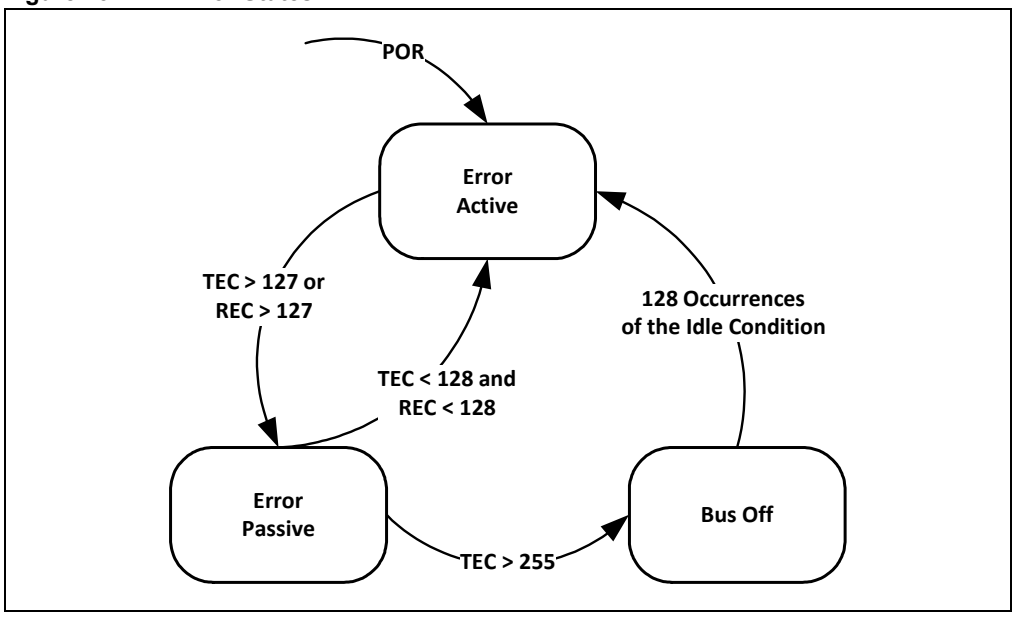
Error detection and Fault confinement are described in the ISO11898-1:2015. CxTRECL contains the error counters, TEC and REC (TERRCNTx, RERRCNTx). CxTRECH contains the error warning and error state bits. TEC and REC increment and decrement according to ISO11898-1:2015 specifications.

Figure 13-1 illustrates the different error states of the CAN FD Protocol Module. The module starts in the error active state. If the TEC or REC exceeds 127, the module transitions to the error passive state. If the TEC exceeds 255, the module will transition to the bus off state.

The module transmits active error frames when in an error active state. It will transmit passive error frames while in an error passive state. When the module is in bus off, the CxTX pin is always driven high and no dominant bits are transmitted.

To avoid the module from transitioning to the error passive state, the module will alert the application when the TEC or REC reaches 96, using the CERRIF interrupt flag (see **Section 12.3.3 "CAN Bus Error Interrupt – CERRIF"**). This allows the application to take action before it enters the error passive state.

Figure 13-1: Error States



The bus diagnostic registers provide additional information about the health of the CAN bus:

- CxBDIAG0L and CxBDIAG0H contain separate error counters for receive/transmit and for Nominal/Data Bit Rates. The counters work differently than the counters in the CxTRECL/H registers. They are simply incremented by one on every error. They are never decremented, but can be cleared by writing '0' to the register.
- CxBDIAG1H keeps track of the kind of error that occurred since the last clearing of the register. The CxBDIAG1L register also contains the error-free message counter. The flags and the counter are cleared by writing '0' to the register.

The error-free message counter, together with the error counters and error flags, can be used to determine the quality of the bus.

13.1 Recovery from Bus Off State

If the TEC exceeds 255, the TXBO (CxTRECH[5]) and CERRIF (CxINTL[13]) bits will be set. The module will go to bus off and start the bus off recovery sequence.

The bus off recovery sequence starts automatically. The module will transition out of the bus off state only after the detection of 128 Idle conditions (see *"ISO11898-1:2015: Bus Off Management"*). The module will set FRESET for all transmit FIFOs when entering the bus off state to ensure that the module does not try to retransmit indefinitely. The application will be notified by CERRIF and has the option to queue new messages for transmission.

The module signals the exit from the bus off state with the CERRIF bit and by setting the TXBOERR bit (CxBDIAG1H[7]). Additionally, CxTRECL/H will be reset.

14.0 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 dsPIC33/PIC24 device families, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the CAN FD Protocol Module include the following:

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

Note: Please visit the Microchip website (www.microchip.com) for additional application notes and code examples for the dsPIC33/PIC24 families of devices.

15.0 REVISION HISTORY

Revision A (February 2018)

This is the initial version of this document.

Revision B (January 2019)

Apart from the modified items given in the following list, this revision shows modified bit representation (e.g., bits<3:0> have been changed to bits[3:0]). This is done to be consistent with documents that were created in the SDL software.

- Sections:
  - Added **Section 5.3.7 “Recommendations for Bit Time Configuration”**.
  - Updated **Section 4.1.4 “Entering and Exiting Disable Mode”**, **Section 5.0 “Configuration”** and **Section 8.0 “Message Filtering”**.
- Figures:
  - Updated Figure 1-2 and Figure 2-3.
- Tables:
  - Updated Table 5-3, Table 6-1 and Table 7-1.
- Registers:
  - Updated Register 3-2, Register 3-1, Register 3-45, Register 3-51, Register 3-50 and Register 5-1.
- Examples:
  - Updated Example 6-1, Example 7-1 and Example 9-1.

Revision C (December 2022)

Organized all registers by low/high bit order for consistency. Changed all register names to represent all low/high register possibilities (i.e., Register 3-2: C**1**CONL has been changed to C**x**CONL).

- Sections:
  - Updated **Section 1.0 “Introduction”**, **Section 3.0 “Control Registers”**, **Section 4.2 “Configuration Mode”**, **Section 5.0 “Configuration”**, **Section 5.3 “CAN FD Bit Time Configuration”**, **Section 5.3.7 “Recommendations for Bit Time Configuration”**, **Section 5.3.8 “Bit Time Configuration Example”**, **Section 5.4 “Message Memory Configuration”**, **Section 6.6 “CxTXREQ Register”**, **Section 11.0 “Time-stamping”** and **Section 13.0 “Error Handling”**.
  - Added **Section 5.4.6 “Calculation of Start Addresses”**.
- Figures:
  - Updated Figure 2-1 and Figure 5-3.
- Tables:
  - Updated Table 5-3, Table 5-4, Table 6-1, Table 7-1 and Table 9-1.
- Registers:
  - Updated Register 3-8, Register 3-7 and Register 3-27.
- Equations:
  - Updated Equation 5-3, Equation 5-4 and Equation 5-11.
- Example
  - Updated Example 6-1, Example 7-1 and Example 9-1.

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable" Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Trademarks

ISBN: 978-1-6683-1242-1

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820