# Section 23. CAN Module

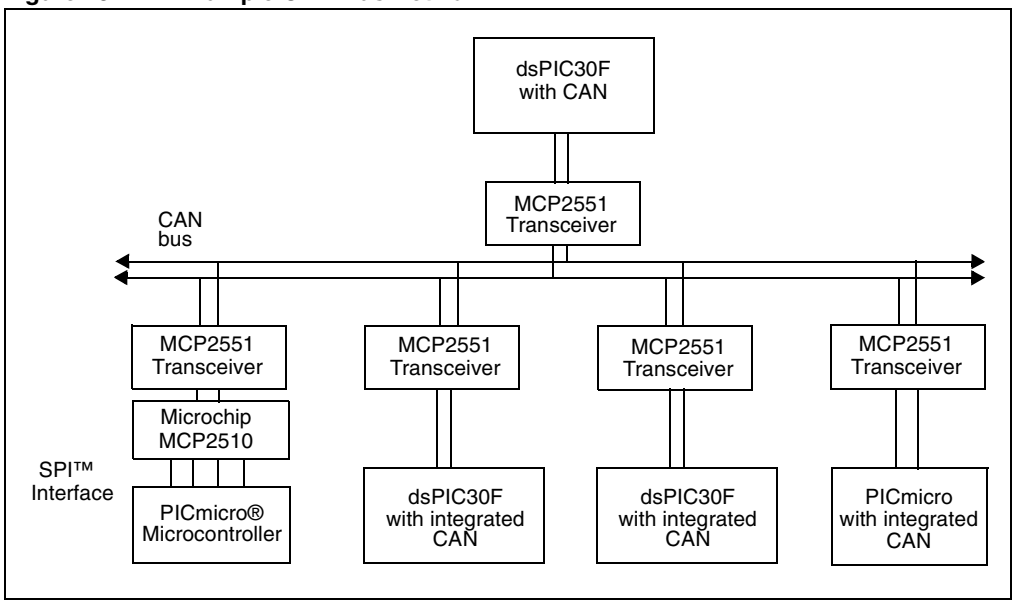
## HIGHLIGHTS

This section of the manual contains the following major topics:

## 23.1 Introduction

The Controller Area Network (CAN) module is a serial interface useful for communicating with other peripherals or microcontroller devices. This interface/protocol was designed to allow communications within noisy environments. Figure 23-1 shows an example CAN bus network.

**Figure 23-1: Example CAN Bus Network**




## 23.2 Control Registers for the CAN Module

There are many registers associated with the CAN module. Descriptions of these registers are grouped into sections. These sections are:

- Control and Status Registers
- Transmit Buffer Registers
- Receive Buffer Registers
- Baud Rate Control Registers
- Interrupt Status and Control Registers

**Note 1:** 'i' in the register identifier denotes the specific CAN module (CAN1 or CAN2).
**2:** 'n' in the register identifier denotes the buffer, filter or mask number.
**3:** 'm' in the register identifier denotes the word number within a particular CAN data field.

### 23.2.1 CAN Control and Status Registers

**Register 23-1: CiCTRL: CAN Module Control and Status Register**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-1 | R/W-0 | R/W-0 |
| TSTAMP | — | CSIDL | ABAT | CANCKS | REQOP<2:0> | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R-1 | R-0 | R-0 | U-0 | R-0 | R-0 | R-0 | U-0 |
| OPMODE<2:0> | | | — | ICODE<2:0> | | | — |
| bit 7 | | | | | | | bit 0 |

bit 15 **TSTAMP:** CAN Message Receive Capture Enable bit
1 = Enable CAN capture
0 = Disable CAN capture
**Note:** TSTAMP is always writable, regardless of CAN module Operating mode.

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

bit 13 **CSIDL:** Stop in IDLE Mode bit
1 = Discontinue CAN module operation when device enters IDLE mode
0 = Continue CAN module operation in IDLE mode

bit 12 **ABAT:** Abort All Pending Transmissions bit
1 = Abort pending transmissions in all Transmit Buffers
0 = No effect
**Note:** Module will clear this bit when all transmissions aborted.

bit 11 **CANCKS:** CAN Master Clock Select bit
1 = FCAN clock is FCY
0 = FCAN clock is 4 FCY

bit 10-8 **REQOP<2:0>:** Request Operation Mode bits
111 = Set Listen All Messages mode
110 = Reserved
101 = Reserved
100 = Set Configuration mode
011 = Set Listen Only mode
010 = Set Loopback mode
001 = Set Disable mode
000 = Set Normal Operation mode

bit 7-5 **OPMODE<2:0>:** Operation Mode bits
**Note:** These bits indicate the current Operating mode of the CAN module. See description for REQOP bits (CiCTRL<10:8>).

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

**Register 23-1: CiCTRL: CAN Module Control and Status Register (Continued)**

bit 3-1 **ICODE<2:0>:** Interrupt Flag Code bits
111 = Wake-up interrupt
110 = RXB0 interrupt
101 = RXB1 interrupt
100 = TXB0 interrupt
011 = TXB1 interrupt
010 = TXB2 interrupt
001 = Error interrupt
000 = No interrupt

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

### 23.2.2 CAN Transmit Buffer Registers

This subsection describes the CAN Transmit Buffer Register and the associated Transmit Buffer Control Registers.

**Register 23-2: CiTXnCON: Transmit Buffer Status and Control Register**

| Upper Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| Lower Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | R-0 | R-0 | R-0 | R/W-0 | U-0 | R/W-0 | R/W-0 |
| — | TXABT | TXLARB | TXERR | TXREQ | — | TXPRI<1:0> | |
| bit 7 | | | | | | | bit 0 |

bit 15-7 **Unimplemented:** Read as '0'

bit 6 **TXABT:** Message Aborted bit
1 = Message was aborted
0 = Message has not been aborted
**Note:** This bit is cleared when TXREQ is set.

bit 5 **TXLARB:** Message Lost Arbitration bit
1 = Message lost arbitration while being sent
0 = Message did not lose arbitration while being sent
**Note:** This bit is cleared when TXREQ is set.

bit 4 **TXERR:** Error Detected During Transmission bit
1 = A bus error occurred while the message was being sent
0 = A bus error did not occur while the message was being sent
**Note:** This bit is cleared when TXREQ is set.

bit 3 **TXREQ:** Message Send Request bit
1 = Request message transmission
0 = Abort message transmission if TXREQ already set, otherwise no effect
**Note:** The bit will automatically clear when the message is successfully sent.

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

bit 1-0 **TXPRI<1:0>:** Message Transmission Priority bits
11 = Highest message priority
10 = High intermediate message priority
01 = Low intermediate message Priority
00 = Lowest message priority

| 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 |

**Register 23-3: CiTXnSID: Transmit Buffer n Standard Identifier**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | U-0 | U-0 |
| SID<10:6> | | | | | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| SID<5:0> | | | | | | SRR | TXIDE |
| bit 7 | | | | | | | bit 0 |

bit 15-11 **SID<10:6>:** Standard Identifier bits

bit 10-8 **Unimplemented:** Read as '0'

bit 7-2 **SID<6:0>:** Standard Identifier bits

bit 1 **SRR:** Substitute Remote Request Control bit
1 = Message will request a remote transmission
0 = Normal message.

bit 0 **TXIDE:** Extended Identifier bit
1 = Message will transmit extended identifier
0 = Message will transmit standard identifier

| 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 |

**Register 23-4: CiTXnEID: Transmit Buffer n Extended Identifier**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | U-0 | U-0 | U-0 | U-0 |
| EID<17:14> | | | | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<13:6> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-12 **EID<17:14>:** Extended Identifier bits 17-14

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

bit 7-0 **EID<13:6>:** Extended Identifier bits 13-6

| 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 |

**Register 23-5: CiTXnDLC: Transmit Buffer n Data Length Control**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<5:0> | | | | | | TXRTR | TXRB1 |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | U-0 | U-0 |
| TXRB0 | DLC<3:0> | | | | — | — | — |
| bit 7 | | | | | | | bit 0 |

bit 15-10 **EID<5:0>:** Extended Identifier bits 5-0

bit 9 **TXRTR:** Remote Transmission Request bit
1 = Message will request a remote transmission
0 = Normal message

bit 8-7 **TXRB<1:0>:** Reserved Bits
**Note:** User must set these bits to '1' according to CAN protocol.

bit 6-3 **DLC<3:0>:** Data Length Code bits

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

**Register 23-6: CiTXnBm: Transmit Buffer n Data Field Word m**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| CTXB<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| CTXB<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-0 **CTXB<15:0>:** Data Field Buffer Word bits (2 bytes)

| 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 |

### 23.2.3 CAN Receive Buffer Registers

This subsection shows the Receive buffer registers with their associated control registers.

**Register 23-7: CiRX0CON: Receive Buffer 0 Status and Control Register**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/C-0 | U-0 | U-0 | U-0 | R-0 | R/W-0 | R/W-0 | R-0 |
| RXFUL | — | — | — | RXRTRRO | DBEN | JTOFF | FILHIT0 |
| bit 7 | | | | | | | bit 0 |

bit 15-8 **Unimplemented:** Read as '0'

bit 7 **RXFUL:** Receive Full Status bit
1 = Receive buffer contains a valid received message
0 = Receive buffer is open to receive a new message
**Note:** This bit is set by the CAN module and should be cleared by software after the buffer is read.

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **RXRTRRO:** Received Remote Transfer Request bit (read only)
1 = Remote Transfer Request was received
0 = Remote Transfer Request not received
**Note:** This bit reflects the status of the last message loaded into Receive Buffer 0.

bit 2 **DBEN:** Receive Buffer 0 Double Buffer Enable bit
1 = Receive Buffer 0 overflow will write to Receive Buffer 1
0 = No Receive Buffer 0 overflow to Receive Buffer 1

bit 1 **JTOFF:** Jump Table Offset bit (read only copy of DBEN)
1 = Allows Jump Table offset between 6 and 7
0 = Allows Jump Table offset between 0 and 1

bit 0 **FILHIT0:** Indicates Which Acceptance Filter Enabled the Message Reception bit
1 = Acceptance Filter 1 (RXF1)
0 = Acceptance Filter 0 (RXF0)
**Note:** This bit reflects the status of the last message loaded into Receive Buffer 0.

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

**Register 23-8: CiRX1CON: Receive Buffer 1 Status and Control Register**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/C-0 | U-0 | U-0 | U-0 | R-0 | R-0 | R-0 | R-0 |
| RXFUL | — | — | — | RXRTRRO | FILHIT<2:0> | | |
| bit 7 | | | | | | | bit 0 |

bit 15-8 **Unimplemented:** Read as '0'

bit 7 **RXFUL:** Receive Full Status bit
1 = Receive buffer contains a valid received message
0 = Receive buffer is open to receive a new message
**Note:** This bit is set by the CAN module and should be cleared by software after the buffer is read.

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **RXRTRRO:** Received Remote Transfer Request bit (read only)
1 = Remote transfer request was received
0 = Remote transfer request not received
**Note:** This bit reflects the status of the last message loaded into Receive Buffer 1.

bit 2-0 **FILHIT<2:0>:** Indicates Which Acceptance Filter Enabled the Message Reception bits
101 = Acceptance filter 5 (RXF5)
100 = Acceptance filter 4 (RXF4)
011 = Acceptance filter 3 (RXF3)
010 = Acceptance filter 2 (RXF2)
001 = Acceptance filter 1 (RXF1) (Only possible when DBEN bit is set)
000 = Acceptance filter 0 (RXF0) (Only possible when DBEN bit is set)

| 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 |

**Register 23-9: CiRXnSID: Receive Buffer n Standard Identifier**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | SID<10:6> | | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| SID<5:0> | | | | | | SRR | RXIDE |
| bit 7 | | | | | | | bit 0 |

bit 15-13 **Unimplemented:** Read as '0'

bit 12-2 **SID<10:0>:** Standard Identifier bits

bit 1 **SRR:** Substitute Remote Request bit (Only when RXIDE = 1)
1 = Remote transfer request occured
0 = No remote transfer request occured

bit 0 **RXIDE:** Extended Identifier Flag bit
1 = Received message is an extended data frame, SID<10:0> are EID<28:18>
0 = Received message is a standard data frame

| 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 |

**Register 23-10: CiRXnEID: Recieve Buffer n Extended Identifier**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | — | EID<17:14> | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<13:6> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-12 **Unimplemented:** Read as '0'

bit 11-0 **EID<17:6>:** Extended Identifier bits 17-6

| 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 |

**Register 23-11: CiRXnBm: Receive Buffer n Data Field Word m**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| CRXB<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| CRXB<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-0 **CRXB<15:0>:** Data Field Buffer Word bits (2 bytes)

| 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 |

**Register 23-12: CiRXnDLC: Recieve Buffer n Data Length Control**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<5:0> | | | | | | RXRTR | RB1 |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | RB0 | DLC<3:0> | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-10 **EID<5:0>:** Extended Identifier bits

bit 9 **RXRTR:** Receive Remote Transmission Request bit
1 = Remote transfer request
0 = No remote transfer request
**Note:** This bit reflects the status of the RTR bit in the last received message.

bit 8 **RB1:** Reserved bit 1
Reserved by CAN Spec and read as '0'

bit 4 **RB0:** Reserved bit 0
Reserved by CAN Spec and read as '0'

bit 3-0 **DLC<3:0>:** Data Length Code bits (Contents of Receive Buffer)

| 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 |

### 23.2.4 Message Acceptance Filters

This subsection describes the Message Acceptance filters.

**Register 23-13: CiRXFnSID: Acceptance Filter n Standard Identifier**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | SID<10:6> | | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | R/W-x |
| SID<5:0> | | | | | | — | EXIDE |
| bit 7 | | | | | | | bit 0 |

bit 15-13 **Unimplemented:** Read as '0'

bit 12-2 **SID<10:0>:** Standard Identifier bits

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

bit 0 **EXIDE:** Extended Identifier Enable bits
If MIDE = 1, then
1 = Enable filter for extended identifier
0 = Enable filter for standard identifier
If MIDE = 0, then EXIDE is don't care

| 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 |

**Register 23-14: CiRXFnEIDH: Acceptance Filter n Extended Identifier High**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | — | EID<17:14> | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<13:6> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-12 **Unimplemented:** Read as '0'

bit 11-0 **EID<17:6>:** Extended Identifier bits 17-6

| 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 |

**Register 23-15: CiRXFnEIDL: Acceptance Filter n Extended Identifier Low**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | U-0 |
| EID<5:0> | | | | | | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

bit 15-10 **EID<5:0>:** Extended Identifier bits

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

### 23.2.5 Acceptance Filter Mask Registers

**Register 23-16: CiRXMnSID: Acceptance Filter Mask n Standard Identifier**

| Upper Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | SID<10:6> | | | | |
| bit 15 | | | | | | | bit 8 |

| Lower Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | R/W-x |
| SID<5:0> | | | | | | — | MIDE |
| bit 7 | | | | | | | bit 0 |

bit 15-13 **Unimplemented:** Read as '0'

bit 12-2 **SID<10:0>:** Standard Identifier Mask bits
1 = Include bit in the filter comparison
0 = Don't include bit in the filter comparison

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

bit 0 **MIDE:** Identifier Mode Selection bit
1 = Match only message types (standard or extended address) as determined by EXIDE bit in filter
0 = Match either standard or extended address message if the filters match

| 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 |

**Register 23-17: CiRXMnEIDH: Acceptance Filter Mask n Extended Identifier High**

| Upper Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x | R/W-x |
| — | — | — | — | EID<17:14> | | | |
| bit 15 | | | | | | | bit 8 |

| Lower Byte: | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| EID<13:6> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-12 **Unimplemented:** Read as '0'

bit 11-0 **EID<17:6>:** Extended Identifier Mask bits 17-6
1 = Include bit in the filter comparison
0 = Don't include bit in the filter comparison

| 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 |

**Register 23-18: CiRXMnEIDL: Acceptance Filter Mask n Extended Identifier Low**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | U-0 | U-0 |
| EID<5:0> | | | | | | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

bit 15-10 **EID<5:0>:** Extended Identifier bits

bit 9-0 **Unimplemented:** Read as ‘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 |

### 23.2.6 CAN Baud Rate Registers

This subsection describes the CAN baud rate registers.

**Register 23-19: CiCFG1: Baud Rate Configuration Register 1**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| SJW<1:0> | | BRP<5:0> | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-8 **Unimplemented:** Read as '0'

bit 7-6 **SJW<1:0>:** Synchronized Jump Width bits
11 = Synchronized jump width time is 4 x TQ
10 = Synchronized jump width time is 3 x TQ
01 = Synchronized jump width time is 2 x TQ
00 = Synchronized jump width time is 1 x TQ

bit 5-0 **BRP<5:0>:** Baud Rate Prescaler bits
11 1111 = TQ = 2 x (BRP + 1)/FCAN = 128/FCAN
.
.
00 0000 = TQ = 2 x (BRP + 1)/FCAN = 2/FCAN

**Note:** FCAN is FCY or 4 FCY, depending on the CANCKS bit setting.

| 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 |

**Register 23-20: CiCFG2: Baud Rate Configuration Register 2**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | R/W-x | U-0 | U-0 | U-0 | R/W-x | R/W-x | R/W-x |
| — | WAKFIL | — | — | — | SEG2PH<2:0> | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x | R/W-x |
| SEG2PHTS | SAM | SEG1PH<2:0> | | | PRSEG<2:0> | | |
| bit 7 | | | | | | | bit 0 |

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

bit 14 **WAKFIL:** Select CAN bus Line Filter for Wake-up bit
1 = Use CAN bus line filter for wake-up
0 = CAN bus line filter is not used for wake-up

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

bit 10-8 **SEG2PH<2:0>:** Phase Buffer Segment 2 bits
111 = length is 8 x TQ
.
.
000 = length is 1 x TQ

bit 7 **SEG2PHTS:** Phase Segment 2 Time Select bit
1 = Freely programmable
0 = Maximum of SEG1PH or information processing time (3 TQ's), whichever is greater

bit 6 **SAM:** Sample of the CAN bus Line bit
1 = Bus line is sampled three times at the sample point
0 = Bus line is sampled once at the sample point

bit 5-3 **SEG1PH<2:0>:** Phase Buffer Segment 1 bits
111 = length is 8 x TQ
.
.
000 = length is 1 x TQ

bit 2-0 **PRSEG<2:0>:** Propagation Time Segment bits
111 = length is 8 x TQ
.
.
000 = length is 1 x TQ

| 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 |

### 23.2.7 CAN Module Error Count Register

This subsection describes the CAN Module Transmission/Reception Error Count register. The various error status flags are present in the CAN Interrupt Flag Register.

**Register 23-21: CiEC: Transmit/Receive Error Count**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| TERRCNT<7:0> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| RERRCNT<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

bit 15-8 **TERRCNT<7:0>:** Transmit Error Count bits

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

| 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 |

### 23.2.8 CAN Interrupt Registers

This subsection documents the CAN Registers which are associated with interrupts.

**Register 23-22: CiINTE: Interrupt Enable Register**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| IVRIE | WAKIE | ERRIE | TX2IE | TX1IE | TX0IE | RX1IE | RX0IE |
| bit 7 | | | | | | | bit 0 |

bit 15-8 **Unimplemented:** Read as '0'

bit 7 **IVRIE:** Invalid Message Received Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 6 **WAKIE:** Bus Wake Up Activity Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 5 **ERRIE:** Error Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 4 **TX2IE:** Transmit Buffer 2 Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 3 **TX1IE:** Transmit Buffer 1 Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 2 **TX0IE:** Transmit Buffer 0 Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 1 **RX1IE:** Receive Buffer 1 Interrupt Enable bit
1 = Enabled
0 = Disabled

bit 0 **RX0IE:** Receive Buffer 0 Interrupt Enable bit
1 = Enabled
0 = Disabled

| 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 |

**Register 23-23: CiINTF: Interrupt Flag Register**

| **Upper Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/C-0 | R/C-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| RX0OVR | RX1OVR | TXBO | TXEP | RXEP | TXWAR | RXWAR | EWARN |
| bit 15 | | | | | | | bit 8 |

| **Lower Byte:** | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
| IVRIF | WAKIF | ERRIF | TX2IF | TX1IF | TX0IF | RX1IF | RX0IF |
| bit 7 | | | | | | | bit 0 |

bit 15 **RX0OVR:** Receive Buffer 0 Overflowed bit
1 = Receive buffer 0 overflowed
0 = Receive buffer 0 not overflowed

bit 14 **RX1OVR:** Receive Buffer 1 Overflowed bit
1 = Receive buffer 1 overflowed
0 = Receive buffer 1 not overflowed

bit 13 **TXBO:** Transmitter in Error State, Bus Off bit
1 = Transmitter in error state, bus off
0 = Transmitter not in error state, bus off

bit 12 **TXEP:** Transmitter in Error State, Bus Passive bit
1 = Transmitter in error state, bus passive
0 = Transmitter not in error state, bus passive

bit 11 **RXEP:** Receiver in Error State, Bus Passive bit
1 = Receiver in error state, bus passive
0 = Receiver not in error state, bus passive

bit 10 **TXWAR:** Transmitter in Error State, Warning bit
1 = Transmitter in error state, warning
0 = Transmitter not in error state, warning

bit 9 **RXWAR:** Receiver in Error State, Warning bit
1 = Receiver in error state, warning
0 = Receiver not in error state, warning

bit 8 **EWARN:** Transmitter or Receiver is in Error State, Warning bit
1 = Transmitter or receiver is in error state, warning
0 = Transmitter and receiver are not in error state

bit 7 **IVRIF:** Invalid Message Received Interrupt Flag bit
1 = Some type of error occurred during reception of the last message
0 = Receive error has not occurred

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

bit 5 **ERRIF:** Error Interrupt Flag bit (multiple sources in CiINTF<15:8> register)
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 4 **TX2IF:** Transmit Buffer 2 Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 3 **TX1IF:** Transmit Buffer 1 Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

**Register 23-23: CiINTF: Interrupt Flag Register (Continued)**

bit 2 **TX0IF:** Transmit Buffer 0 Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 1 **RX1IF:** Receive Buffer 1 Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

bit 0 **RX0IF:** Receive Buffer 0 Interrupt Flag bit
1 = Interrupt request has occurred
0 = Interrupt request has not occurred

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

**Table 23-1: CAN1 Register Map**

| File Name | ADR | Bit | | | | | | | | | | | | | | | | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| C1RXF0SID | 300 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF0EIDH | 302 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF0EIDL | 304 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 306 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXF1SID | 308 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF1EIDH | 30A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF1EIDL | 30C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 30E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXF2SID | 310 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF2EIDH | 312 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF2EIDL | 314 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 316 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXF3SID | 318 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF3EIDH | 31A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF3EIDL | 31C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 31E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXF4SID | 320 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF4EIDH | 322 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF4EIDL | 324 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 326 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXF5SID | 328 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C1RXF5EIDH | 32A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXF5EIDL | 32C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 32E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXM0SID | 330 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | MIDE | xxxx |
| C1RXM0EIDH | 332 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXM0EIDL | 334 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 336 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C1RXM1SID | 338 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | MIDE | xxxx |
| C1RXM1EIDH | 33A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RXM1EIDL | 33C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 33E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |

**Table 23-1: CAN1 Register Map (Continued)**

| File Name | ADR | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1TX2SID | 340 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C1TX2EID | 342 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C1TX2DLC | 342 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C1TX2D01 | 346 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C1TX2D23 | 348 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C1TX2D45 | 34A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C1TX2D67 | 34C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C1TX2CON | 34E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |
| C1TX1SID | 350 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C1TX1EID | 352 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C1TX1DLC | 352 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C1TX1D01 | 356 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C1TX1D23 | 358 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C1TX1D45 | 35A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C1TX1D67 | 35C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C1TX1CON | 35E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |
| C1TX0SID | 360 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C1TX0EID | 362 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C1TX0DLC | 362 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C1TX0D01 | 366 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C1TX0D23 | 368 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C1TX0D45 | 36A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C1TX0D67 | 36C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C1TX0CON | 36E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |

**Table 23-1: CAN1 Register Map (Continued)**

| File Name | ADR | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C1RX1SID | 370 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | SRR | RX IDE | xxxx |
| C1RX1EID | 372 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RX1DLC | 374 | EID<0:5> | | | | | | RX RTR | RX RB1 | — | — | — | RX RB0 | DLC[3:0] | | | | xxxx |
| C1RX1D01 | 376 | Receive Buffer 1 Byte 1 | | | | | | | | Receive Buffer 1 Byte 0 | | | | | | | | xxxx |
| C1RX1D23 | 378 | Receive Buffer 1 Byte 3 | | | | | | | | Receive Buffer 1 Byte 2 | | | | | | | | xxxx |
| C1RX1D45 | 37A | Receive Buffer 1 Byte 5 | | | | | | | | Receive Buffer 1 Byte 4 | | | | | | | | xxxx |
| C1RX1D67 | 37C | Receive Buffer 1 Byte 7 | | | | | | | | Receive Buffer 1 Byte 6 | | | | | | | | xxxx |
| C1RX1CON | 37E | — | — | — | — | — | — | — | — | RX FUL | — | — | RX ERR | RX RTR R0 | FILHIT[2:0] | | | 0000 |
| C1RX1SID | 380 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | SRR | RX IDE | xxxx |
| C1RX1EID | 382 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C1RX1DLC | 384 | EID<0:5> | | | | | | RX RTR | RX RB1 | — | — | — | RX RB0 | DLC[3:0] | | | | xxxx |
| C1RX0D01 | 386 | Receive Buffer 0 Byte 1 | | | | | | | | Receive Buffer 0 Byte 0 | | | | | | | | xxxx |
| C1RX0D23 | 388 | Receive Buffer 0 Byte 3 | | | | | | | | Receive Buffer 0 Byte 2 | | | | | | | | xxxx |
| C1RX0D45 | 38A | Receive Buffer 0 Byte 5 | | | | | | | | Receive Buffer 0 Byte 4 | | | | | | | | xxxx |
| C1RX0D67 | 38C | Receive Buffer 0 Byte 7 | | | | | | | | Receive Buffer 0 Byte 6 | | | | | | | | xxxx |
| C1RX0CON | 38E | — | — | — | — | — | — | — | — | RX FUL | — | — | RX ERR | RX RTR R0 | RXB0 DBEN | JTOFF | FIL HIT 0 | 0000 |
| C1CTRL | 390 | CAN CAP | CAN FRZ | CAN SIDL | ABAT | CAN CKS | REQOP[2:0] | | | OPMODE[2:0] | | | — | ICODE[2:0] | | | — | 0480 |
| C1CFG1 | 392 | — | — | — | — | — | — | — | — | SJW[1:0]S | | BRP[5:0] | | | | | | 0000 |
| C1CFG2 | 394 | — | WAK FIL | — | — | — | SEG2PH[2:0] | | | SEG2 PHTS | SAM | SEG1PH[2:0] | | | PRSEG[2:0] | | | 0000 |
| C1INTF | 396 | RXB0 OVR | RXB1 OVR | TXBO | TXBP | RXBP | TX WARN | RX WARN | E WARN | IVR IF | WAK IF | ERR IF | TXB2 IF | TXB1 IF | TXB0 IF | RXB1 IF | RXB0 IF | 0000 |
| C1INTE | 398 | — | — | — | — | — | — | — | — | IVR IE | WAK IE | ERR IE | TXB2 IE | TXB1 IE | TXB0 IE | RXB1 IE | RXB0 IE | 0000 |
| C1TREC | 39A | Transmit Error Counter | | | | | | | | Receive Error Counter | | | | | | | | 0000 |
| Reserved | 39C 3FE | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |

Legend: x = Unknown

**Table 23-2: CAN2 Register Map**

| File Name | ADR | Bit | | | | | | | | | | | | | | | | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| C2RXF0SID | 400 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF0EIDH | 402 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF0EIDL | 404 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 406 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXF1SID | 408 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF1EIDH | 40A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF1EIDL | 40C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 40E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXF2SID | 410 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF2EIDH | 412 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF2EIDL | 414 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 416 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXF3SIDH | 418 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF3EID | 41A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF3EIDL | 41C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 41E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXF4SID | 420 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF4EIDH | 422 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF4EIDL | 424 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 426 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXF5SID | 428 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | EXIDE | xxxx |
| C2RXF5EIDH | 42A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXF5EIDL | 42C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 42E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXM0SID | 430 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | MIDE | xxxx |
| C2RXM0EIDH | 432 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXM0EIDL | 434 | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 436 | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |
| C2RXM1SID | 438 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | — | MIDE | xxxx |
| C2RXM1EIDH | 43A | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RXM1EIDL | 43C | EID<5:0> | | | | | | — | — | — | — | — | — | — | — | — | — | xxxx |
| unused | 43E | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |

**Table 23-2: CAN2 Register Map (Continued)**

| File Name | ADR | Bit | | | | | | | | | | | | | | | | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| C2TX2SID | 440 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C2TX2EID | 442 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C2TX2DLC | 442 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C2TX2D01 | 446 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C2TX2D23 | 448 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C2TX2D45 | 44A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C2TX2D67 | 44C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C2TX2CON | 44E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |
| C2TX1SID | 450 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C2TX1EID | 452 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C2TX1DLC | 352 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C2TX1D01 | 456 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C2TX1D23 | 458 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C2TX1D45 | 45A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C2TX1D67 | 45C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C2TX1CON | 45E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |
| C2TX0SID | 460 | SID<10:6> | | | | | — | — | — | SID<5:0> | | | | | | SRR | TX IDE | xxxx |
| C2TX0EID | 462 | EID<17:14> | | | | — | — | — | — | EID<13:6> | | | | | | | | xxxx |
| C2TX0DLC | 462 | EID<5:0> | | | | | | TX RTR | TX RB1 | TX RB0 | DLC<3:0> | | | | — | — | — | xxxx |
| C2TX0D01 | 466 | Transmit Buffer 0 Byte 1 | | | | | | | | Transmit Buffer 0 Byte 0 | | | | | | | | xxxx |
| C2TX0D23 | 468 | Transmit Buffer 0 Byte 3 | | | | | | | | Transmit Buffer 0 Byte 2 | | | | | | | | xxxx |
| C2TX0D45 | 46A | Transmit Buffer 0 Byte 5 | | | | | | | | Transmit Buffer 0 Byte 4 | | | | | | | | xxxx |
| C2TX0D67 | 46C | Transmit Buffer 0 Byte 7 | | | | | | | | Transmit Buffer 0 Byte 6 | | | | | | | | xxxx |
| C2TX0CON | 46E | — | — | — | — | — | — | — | — | — | TX ABT | TX LARB | TX ERR | TX REQ | — | TXPRI[1:0] | | 0000 |

**Table 23-2: CAN2 Register Map (Continued)**

| File Name | ADR | Bit | | | | | | | | | | | | | | | | RESET |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| C2RX1SID | 470 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | SRR | RX IDE | xxxx |
| C2RX1EID | 472 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RX1DLC | 474 | EID<0:5> | | | | | | RX RTR | RX RB1 | — | — | — | RX RB0 | DLC[3:0] | | | | xxxx |
| C2RX1D01 | 476 | Receive Buffer 1 Byte 1 | | | | | | | | Receive Buffer 1 Byte 0 | | | | | | | | xxxx |
| C2RX1D23 | 478 | Receive Buffer 1 Byte 3 | | | | | | | | Receive Buffer 1 Byte 2 | | | | | | | | xxxx |
| C2RX1D45 | 47A | Receive Buffer 1 Byte 5 | | | | | | | | Receive Buffer 1 Byte 4 | | | | | | | | xxxx |
| C2RX1D67 | 47C | Receive Buffer 1 Byte 7 | | | | | | | | Receive Buffer 1 Byte 6 | | | | | | | | xxxx |
| C2RX1CON | 47E | — | — | — | — | — | — | — | — | RX FUL | — | — | RX ERR | RX RTR R0 | FILHIT[2:0] | | | 0000 |
| C2RX1SID | 480 | — | — | — | SID<10:6> | | | | | SID<5:0> | | | | | | SRR | RX IDE | xxxx |
| C2RX1EID | 482 | — | — | — | — | EID<17:14> | | | | EID<13:6> | | | | | | | | xxxx |
| C2RX1DLC | 484 | EID<0:5> | | | | | | RX RTR | RX RB1 | — | — | — | RX RB0 | DLC[3:0] | | | | xxxx |
| C2RX0D01 | 486 | Receive Buffer 0 Byte 1 | | | | | | | | Receive Buffer 0 Byte 0 | | | | | | | | xxxx |
| C2RX0D23 | 488 | Receive Buffer 0 Byte 3 | | | | | | | | Receive Buffer 0 Byte 2 | | | | | | | | xxxx |
| C2RX0D45 | 48A | Receive Buffer 0 Byte 5 | | | | | | | | Receive Buffer 0 Byte 4 | | | | | | | | xxxx |
| C2RX0D67 | 48C | Receive Buffer 0 Byte 7 | | | | | | | | Receive Buffer 0 Byte 6 | | | | | | | | xxxx |
| C2RX0CON | 48E | — | — | — | — | — | — | — | — | RX FUL | — | — | RX ERR | RX RTR R0 | RXB0 DBEN | JTOFF | FIL HIT 0 | 0000 |
| C2CTRL | 490 | CAN CAP | CAN FRZ | CAN SIDL | ABAT | CAN CKS | REQOP[2:0] | | | OPMODE[2:0] | | | — | ICODE[2:0] | | | — | 0480 |
| C2CFG1 | 492 | — | — | — | — | — | — | — | — | SJW[1:0]S | | BRP[5:0] | | | | | | 0000 |
| C2CFG2 | 494 | — | WAK FIL | — | — | — | SEG2PH[2:0] | | | SEG2 PHTS | SAM | SEG1PH[2:0] | | | PRSEG[2:0] | | | 0000 |
| C2INTF | 496 | RXB0 OVR | RXB1 OVR | TXBO | TXBP | RXBP | TX WARN | RX WARN | E WARN | IVR IF | WAK IF | ERR IF | TXB2 IF | TXB1 IF | TXB0 IF | RXB1 IF | RXB0 IF | 0000 |
| C2INTE | 498 | — | — | — | — | — | — | — | — | IVR IE | WAK IE | ERR IE | TXB2 IE | TXB1 IE | TXB0 IE | RXB1 IE | RXB0 IE | 0000 |
| C2TREC | 49A | Transmit Error Counter | | | | | | | | Receive Error Counter | | | | | | | | 0000 |
| Reserved | 49C 4FE | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | — | xxxx |

Legend: x = Unknown

### 23.3 CAN Module Features

The CAN module is a communication controller implementing the CAN 2.0A/B protocol as defined in the BOSCH specification. The module will support CAN 1.2, CAN 2.0A, CAN 2.0B Passive and CAN 2.0B Active versions of the protocol. The module implementation is a Full CAN system.

The module features are as follows:

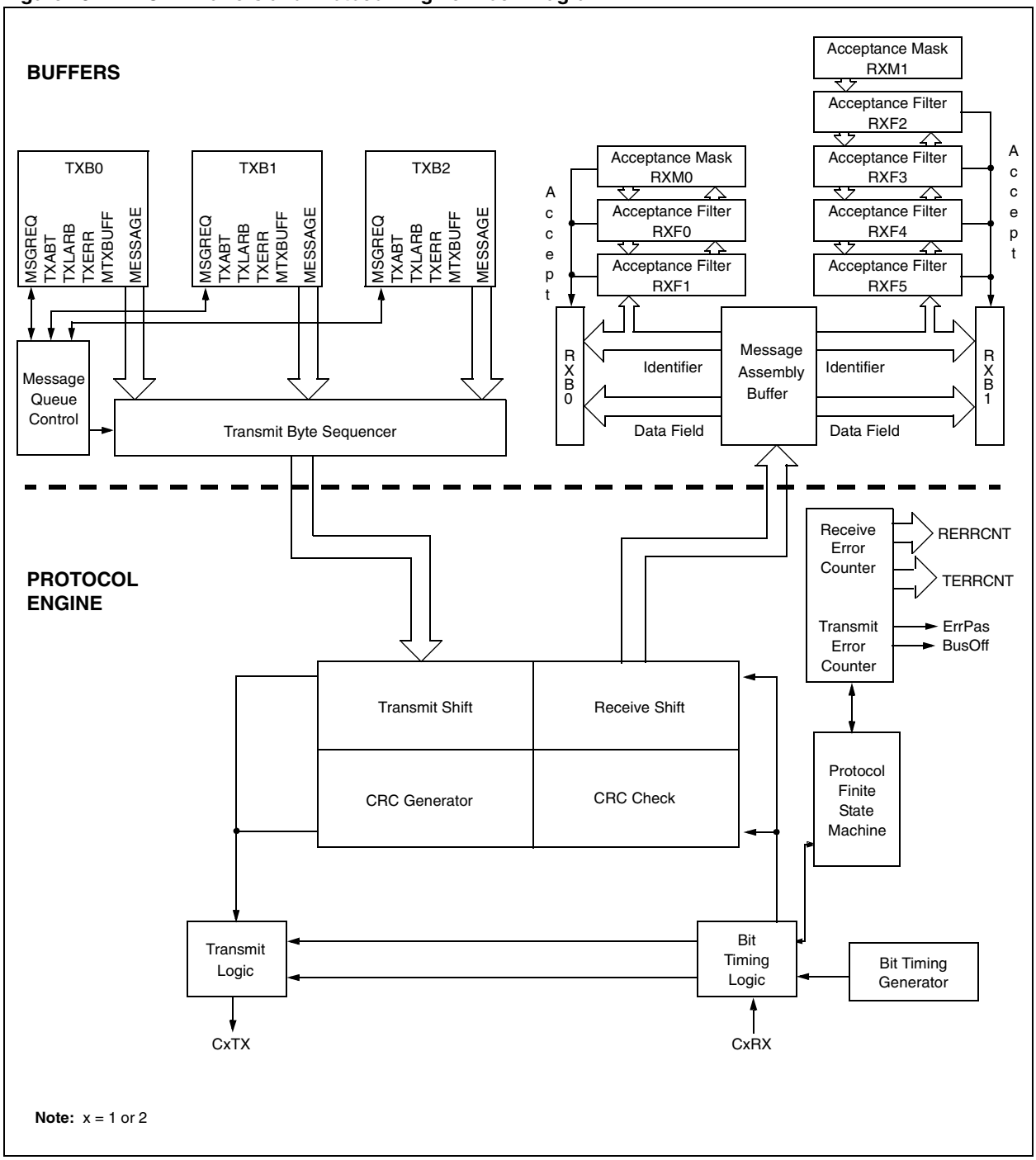
- Implementation of the CAN protocol CAN 1.2, CAN 2.0A and CAN 2.0B
- Standard and extended data frames
- Data length from 0-8 bytes
- Programmable bit rate up to 1 Mbit/sec
- Support for remote data frames
- Double buffered receiver with two prioritized received message storage buffers
- Six full (standard/extended identifier) acceptance filters, 2 associated with the high priority receive buffer, and 4 associated with the low priority receive buffer
- Two full acceptance filter masks, one each associated with the high and low priority receive buffers
- Three Transmit Buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time-stamping and network synchronization
- Low power SLEEP mode

## 23.4 CAN Module Implementation

The CAN bus module consists of a Protocol Engine and message buffering and control. The Protocol Engine can best be understood by defining the types of data frames to be transmitted and received by the module. These blocks are shown in Figure 23-2.

**Figure 23-2: CAN Buffers and Protocol Engine Block Diagram**



### 23.4.1 CAN Message Formats

The CAN protocol engine handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the two receive registers.

The CAN Module supports the following frame types:

- Standard Data Frame
- Extended Data Frame
- Remote Frame
- Error Frame
- Overload Frame
- Interframe Space

#### 23.4.1.1 Standard Data Frame

A standard data frame is generated by a node when the node wishes to transmit data. The standard CAN data frame is shown in Figure 23-3. In common with all other frames, the frame begins with a Start-Of-Frame bit (SOF - dominant state) for hard synchronization of all nodes.

The SOF is followed by the Arbitration field consisting of 12 bits, the 11-bit identifier (reflecting the contents and priority of the message) and the RTR bit (Remote Transmission Request bit). The RTR bit is used to distinguish a data frame (RTR - dominant) from a remote frame.

The next field is the Control field, consisting of 6 bits. The first bit of this field is called the Identifier Extension (IDE) bit and is at dominant state to specify that the frame is a standard frame. The following bit is reserved by the CAN protocol, RB0, and defined as a dominant bit. The remaining 4 bits of the Control field are the Data Length Code (DLC) and specify the number of bytes of data contained in the message.

The data being sent follows in the Data field which is of the length defined by the DLC above (0-8 bytes).

The Cyclic Redundancy Check (CRC) field follows and is used to detect possible transmission errors. The CRC field consists of a 15-bit CRC sequence and a delimiter bit. The message is completed by the End-Of-Frame (EOF) field, which consists of seven recessive bits with no bit-stuffing.

The final field is the Acknowledge field. During the ACK Slot bit the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not). The recessive Acknowledge Delimiter completes the Acknowledge Slot and may not be overwritten by a dominant bit, except when an error frame occurs.

#### 23.4.1.2 Extended Data Frame

In the extended CAN data frame, shown in Figure 23-4, the Start-Of-Frame bit (SOF) is followed by the Arbitration Field consisting of 38 bits. The first 11 bits are the 11 Most Significant bits of the 29-bit identifier (“Base-ID”). These 11 bits are followed by the Substitute Remote Request bit (SRR), which is transmitted as recessive. The SRR is followed by the IDE bit which is recessive to denote that the frame is an extended CAN frame. It should be noted from this, that if arbitration remains unresolved after transmission of the first 11 bits of the identifier, and one of the nodes involved in arbitration is sending a standard CAN frame (11-bit identifier), then the standard CAN frame will win arbitration due to the assertion of a dominant IDE bit. Also, the SRR bit in an extended CAN frame must be recessive to allow the assertion of a dominant RTR bit by a node that is sending a standard CAN remote frame. The SRR and IDE bits are followed by the remaining 18 bits of the identifier (“ID-Extension”) and a dominant Remote Transmission Request bit.

To enable standard and extended frames to be sent across a shared network, it is necessary to split the 29-bit extended message identifier into 11-bit (Most Significant) and 18-bit (Least Significant) sections. This split ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

The next field is the Control field, consisting of 6 bits. The first 2 bits of this field are reserved and are at dominant state. The remaining 4 bits of the Control field are the Data Length Code (DLC) and specify the number of data bytes.

The remaining portion of the frame (Data field, CRC field, Acknowledge field, End-Of-Frame and intermission) is constructed in the same way as for a standard data frame.

#### 23.4.1.3 Remote Frame

A data transmission is usually performed on an autonomous basis with the data source node (e.g., a sensor sending out a data frame). It is possible however for a destination node to request the data from the source. For this purpose, the destination node sends a “remote frame” with an identifier that matches the identifier of the required data frame. The appropriate data source node will then send a data frame as a response to this remote request.

There are two differences between a remote frame and a data frame, shown in Figure 23-5. First, the RTR bit is at the recessive state and second there is no Data field. In the very unlikely event of a data frame and a remote frame with the same identifier being transmitted at the same time, the data frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the remote frame receives the desired data immediately.

#### 23.4.1.4 The Error Frame

An error frame is generated by any node that detects a bus error. An error frame, shown in Figure 23-6, consists of 2 fields, an error flag field followed by an Error Delimiter field. The Error Delimiter consists of 8 recessive bits and allows the bus nodes to restart bus communications cleanly after an error. There are two forms of error flag fields. The form of the error flag field depends on the error status of the node that detects the error.

If an error-active node detects a bus error then the node interrupts transmission of the current message by generating an active error flag. The active error flag is composed of six consecutive dominant bits. This bit sequence actively violates the bit-stuffing rule. All other stations recognize the resulting bit-stuffing error and in turn generate error frames themselves, called Error Echo Flags. The error flag field therefore consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the error frame. After completion of the error frame, bus activity retains to normal and the interrupted node attempts to resend the aborted message.

If an error passive node detects a bus error then the node transmits an Error Passive flag followed, again, by the Error Delimiter field. The Error Passive flag consists of six consecutive recessive bits. From this it follows that, unless the bus error is detected by the transmitting node or other error active receiver that is actually transmitting, the transmission of an error frame by an error passive node will not affect any other node on the network. If the bus master node generates an error passive flag then this may cause other nodes to generate error frames due to the resulting bit-stuffing violation. After transmission of an error frame, an error passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

#### 23.4.1.5 The Overload Frame

An overload frame, shown in Figure 23-7, has the same format as an active error frame. An overload frame, however can only be generated during Interframe Space. This way, an overload frame can be differentiated from an error frame (an error frame is sent during the transmission of a message). The overload frame consists of 2 fields, an overload flag followed by an Overload Delimiter. The overload flag consists of six dominant bits followed by overload flags generated by other nodes (as for active error flag, again giving a maximum of twelve dominant bits). The Overload Delimiter consists of eight recessive bits. An overload frame can be generated by a node as a result of 2 conditions. First, the node detects a dominant bit during Interframe Space which is an illegal condition. Second, due to internal conditions, the node is not yet able to start reception of the next message. A node may generate a maximum of 2 sequential overload frames to delay the start of the next message.

#### 23.4.1.6 The Interframe Space

Interframe Space separates a proceeding frame (of whatever type) from a following data or remote frame. Interframe Space is composed of at least 3 recessive bits, called the intermission. This is provided to allow nodes time for internal processing of the message by receiving nodes before the start of the next message frame. After the intermission, the bus line remains in the recessive state (bus idle) until the next transmission starts.

If the transmitting node is in the error passive state, an additional 8 recessive bit times will be inserted in the Interframe Space before any other message is transmitted by that node. This time period is called the Suspend Transmit field. The Suspend Transmit field allows additional delay time for other transmitting nodes to take control of the bus.

**Figure 23-3: Standard Data Frame**

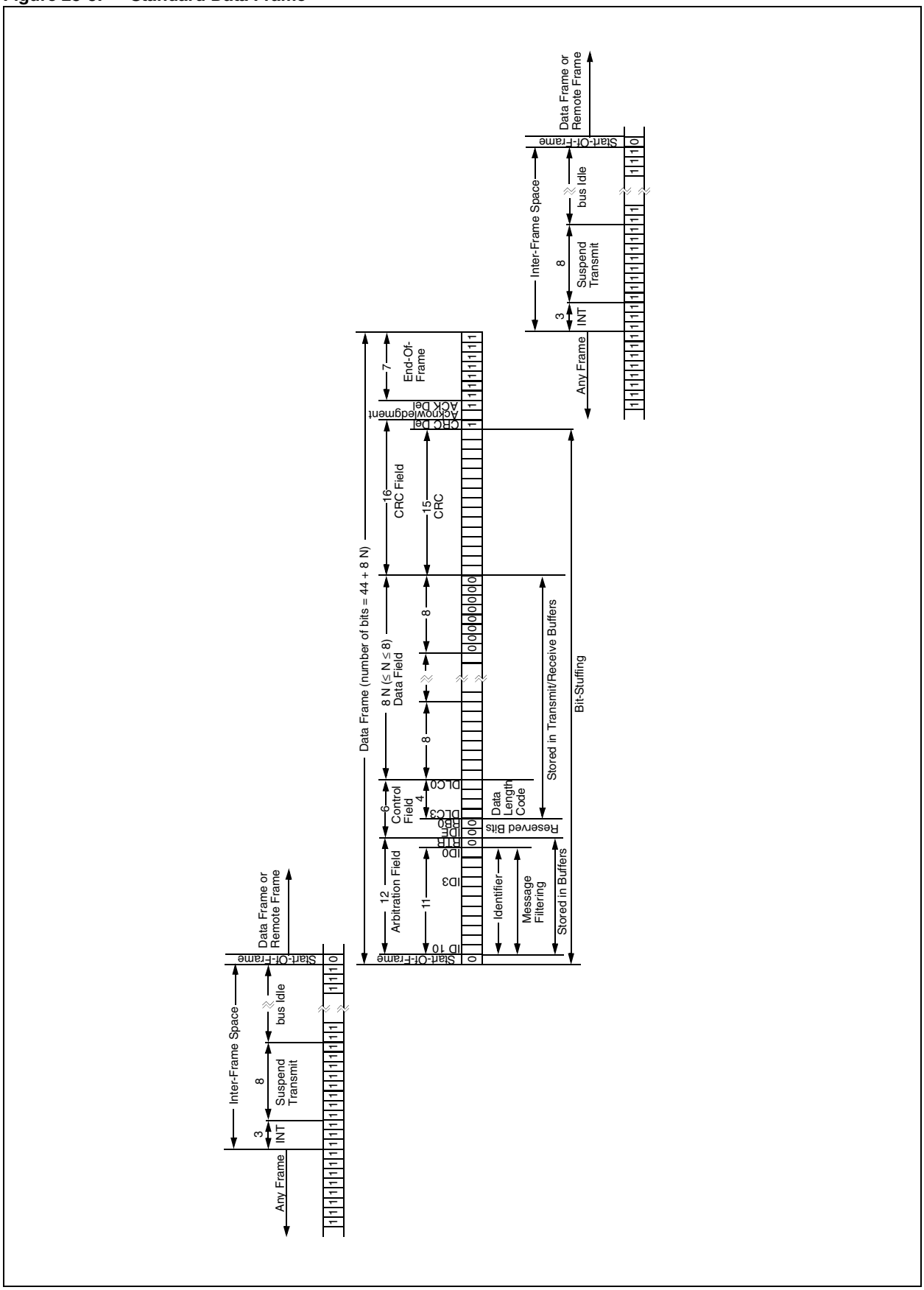


**Figure 23-4: Extended Data Format**

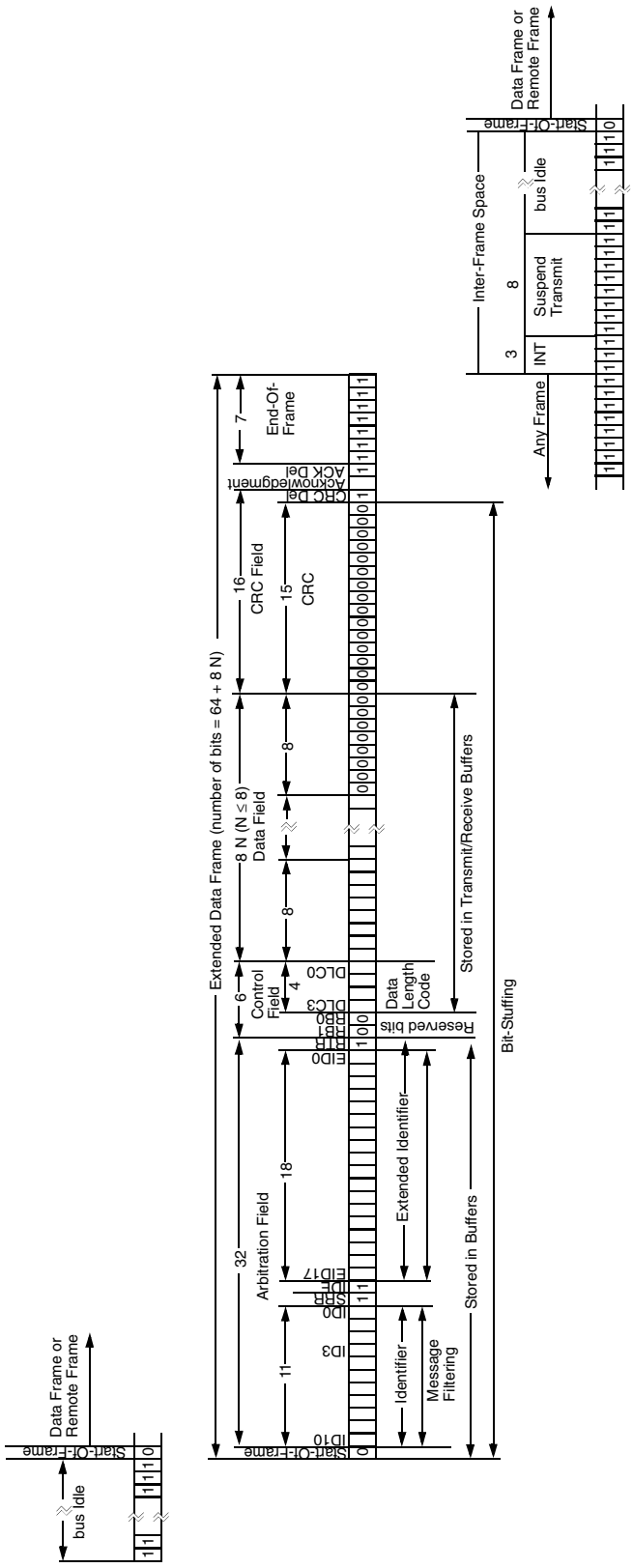


**Figure 23-5: Remote Data Frame**

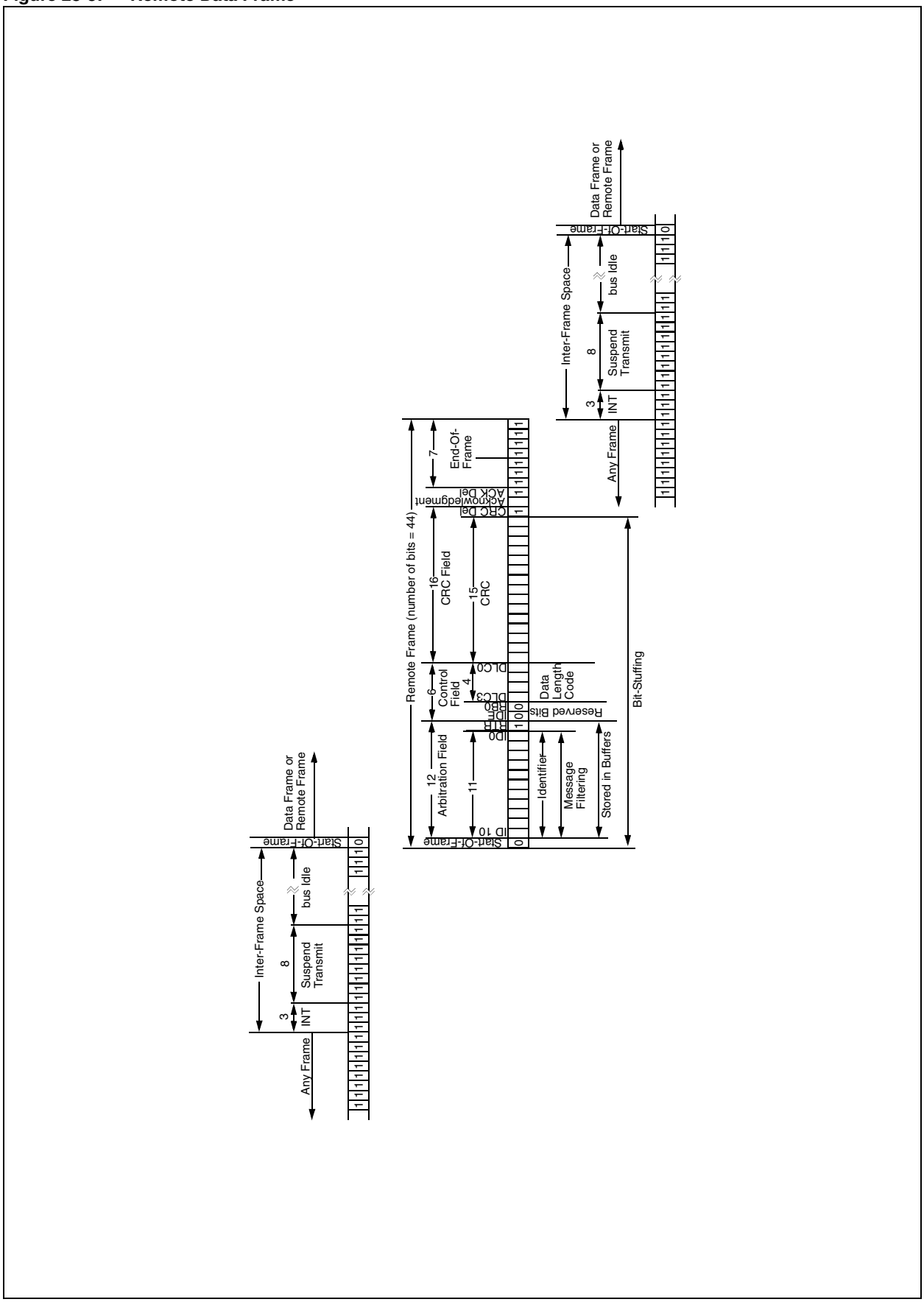


**Figure 23-6: Error Frame**

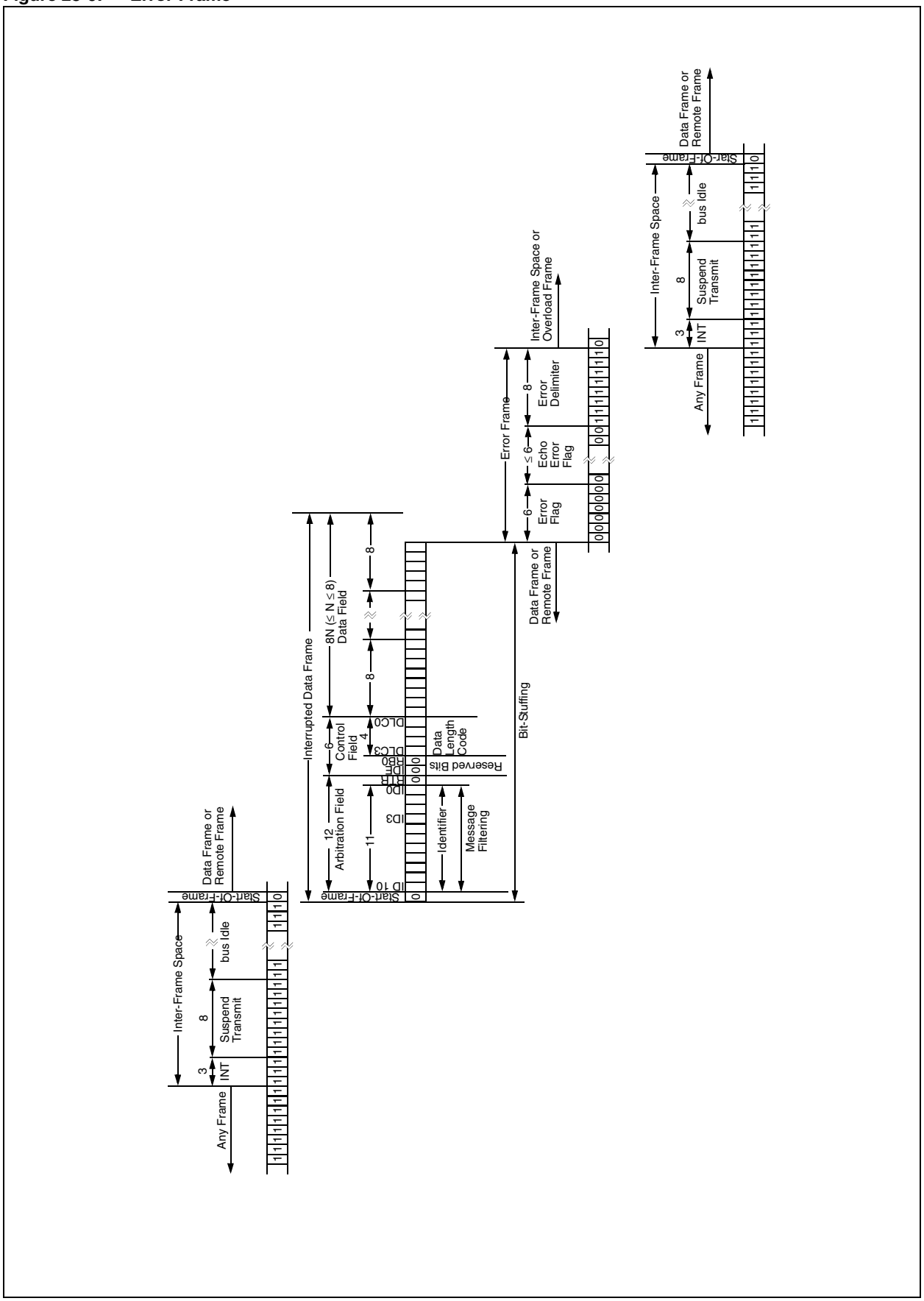
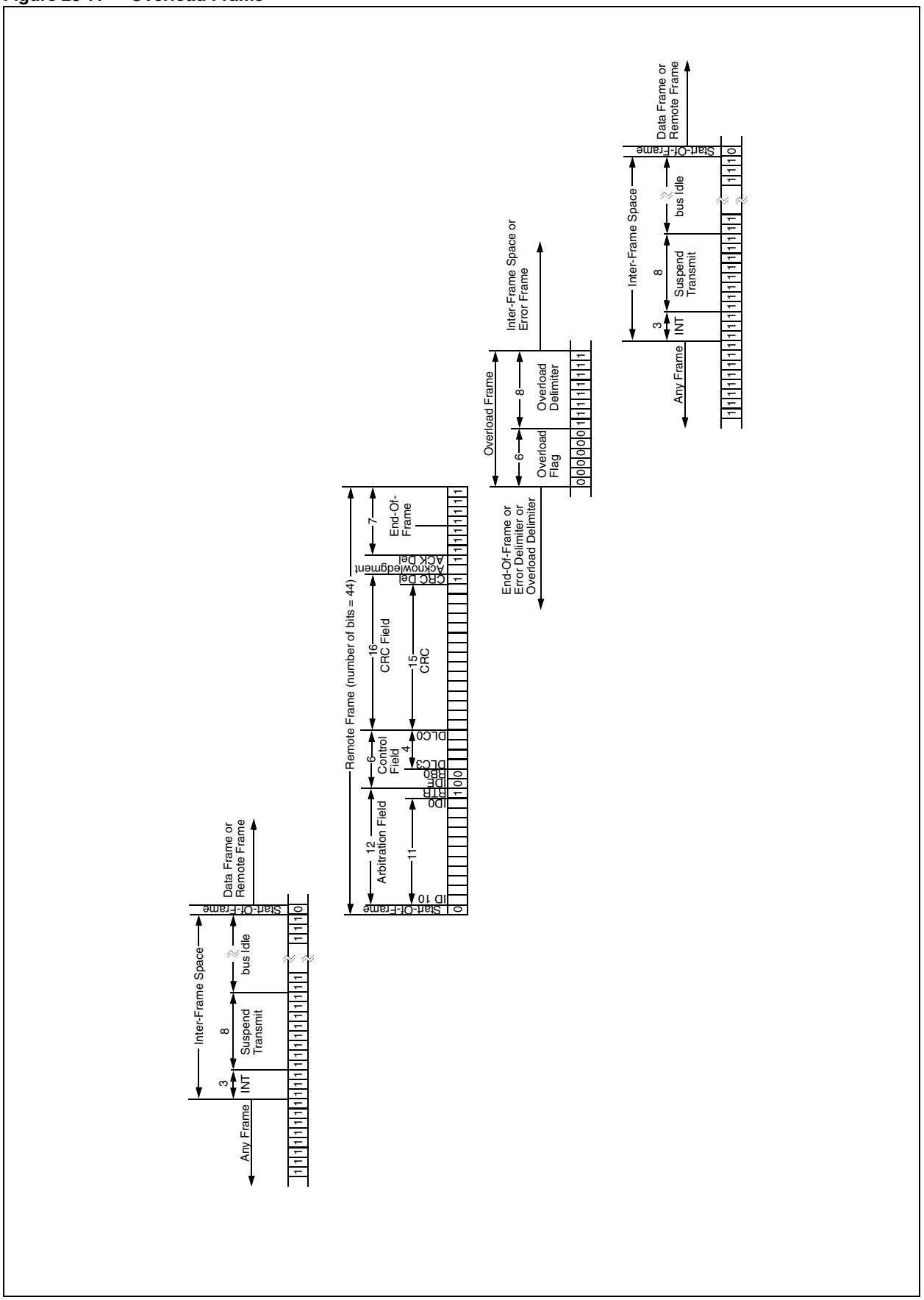


**Figure 23-7: Overload Frame**



## 23.5 CAN Module Operation Modes

The CAN Module can operate in one of several Operation modes selected by the user. These modes include:

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

Modes are requested by setting the REQOP<2:0> bits (CiCTRL<10:8>). Entry into a mode is acknowledged by monitoring the OPMODE<2:0> bits (CiCTRL<7:5>). The module does not change the mode and the OPMODE bits until a change in mode is acceptable, generally during bus idle time which is defined as at least 11 consecutive recessive bits.

### 23.5.1 Normal Operation Mode

Normal Operating mode is selected when REQOP<2:0> = '000'. In this mode, the module is activated, the I/O pins will assume the CAN bus functions. The module will transmit and receive CAN bus messages as described in subsequent sections.

### 23.5.2 Disable Mode

In Disable mode, the module will not transmit or receive. The module has the ability to set the WAKIF bit due to bus activity, however any pending interrupts will remain and the error counters will retain their value.

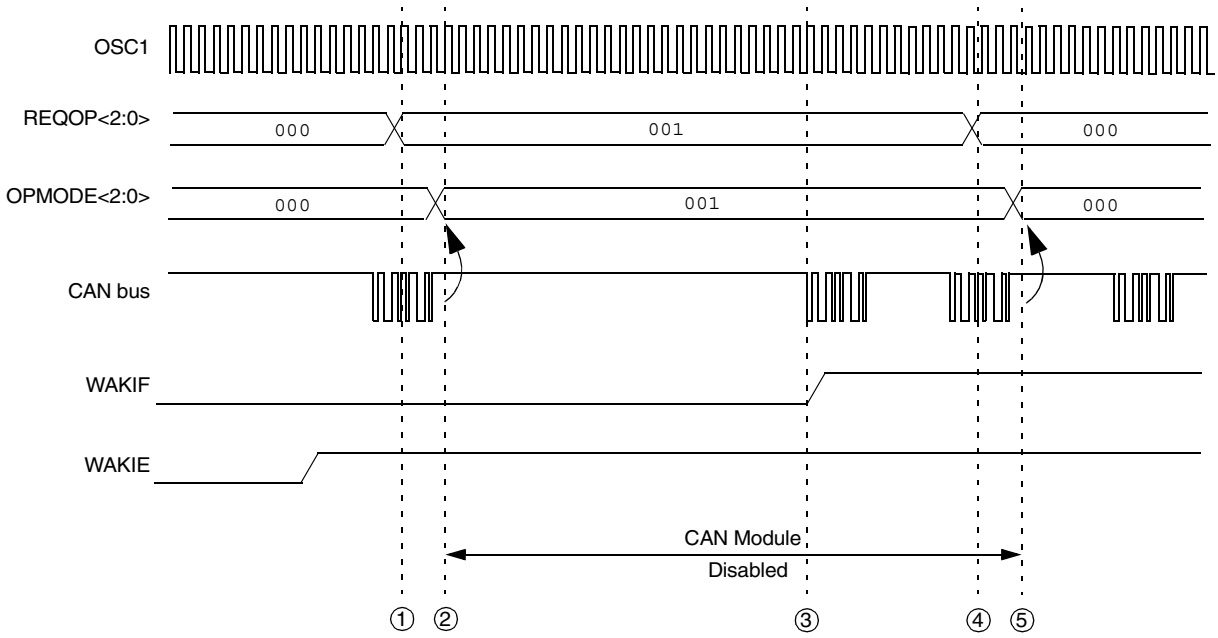
If the REQOP<2:0> bits (CiCTRL<10:8>) = '001', the module will enter the Module Disable mode. This mode is similar to disabling other peripheral modules by turning off the module enables. This causes the module internal clock to stop unless the module is active (i.e., receiving or transmitting a message). If the module is active, the module will wait for 11 recessive bits on the CAN bus, detect that condition as an idle bus, then accept the module disable command. When the OPMODE<2:0> bits (CiCTRL<7:5>) = '001', this indicates that the module successfully entered Module Disable mode (see Figure 23-8).

The WAKIF interrupt is the only module interrupt that is still active in the Module Disable mode. If the WAKIE bit (CiINTE<6>) is set, the processor will receive an interrupt whenever the CAN bus detects a dominant state, as occurs with a Start-Of-Frame (SOF).

The I/O pins will revert to normal I/O function when the module is in the Module Disable mode.

**Figure 23-8: Entering and Exiting Module Disable Mode**




① - Processor writes REQOP<2:0> while module receiving/transmitting message. Module continues with CAN message.
② - Module detects 11 recessive bits. Module acknowledges Disable mode and sets OPMODE<2:0> bits. Module disables.
③ - CAN bus message will set WAKIF bit. If WAKIE = '1', processor will vector to the interrupt address. CAN message ignored.
④ - Processor writes REQOP<2:0> during CAN bus activity. Module waits for 11 recessive bits before accepting activate.
⑤ - Module detects 11 recessive bits. Module acknowledges Normal mode and sets OPMODE<2:0> bits. Module activates.

### 23.5.3 Loopback Mode

If the Loopback mode is activated, the module will connect the internal transmit signal to the internal receive signal at the module boundary. The transmit and receive pins revert to their PORT I/O function.

The transmitter will receive an acknowledge for its sent messages. Special hardware will generate an acknowledge for the transmitter.

### 23.5.4 Listen Only Mode

Listen Only mode and Loopback modes are special cases of Normal Operation mode to allow system debug. If the Listen Only mode is activated, the module on the CAN bus is passive. The transmitter buffers revert to the PORT I/O function. The receive pins remain as inputs to the CAN module. For the receiver, 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. To use this, 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. This mode is also useful as a bus monitor without influencing the data traffic.

### 23.5.5 Configuration Mode

In the Configuration mode, the module will not transmit or receive. The error counters are cleared and the interrupt flags remain unchanged. The programmer will have access to configuration registers that are access restricted in other modes.

After a device RESET the CAN module is in the Configuration mode (OPMODE<2:0> = '100'). The error counters are cleared and all registers contain the RESET values. It should be ensured that the initialization is performed before REQOP<2> bit is cleared.

The CAN module has to be initialized before its activation. This is only possible if the module is in the Configuration mode. The Configuration mode is requested by setting the REQOP<2> bit. Only when the Status bit OPMODE<2> has a high level, the initialization can be performed. Afterwards the configuration registers and the acceptance mask registers and the acceptance filter registers can be written. The module is activated by clearing the control bits REQOP<2:0>.

The module will protect the user from accidentally violating the CAN protocol through programming errors. All registers which control the configuration of the module can not be modified while the module is on-line. The CAN module will not be allowed to enter the Configuration mode while a transmission is taking place. The Configuration mode serves as a lock to protect the following registers.

- All Module Control Registers
- Baud Rate and Interrupt Configuration Registers
- Bus Timing Registers
- Identifier Acceptance Filter Registers
- Identifier Acceptance Mask Registers

### 23.5.6 Listen All Messages Mode

Listen All Messages mode is a special case of Normal Operation mode to allow system debug. If the Listen All Messages mode is activated, the module on the CAN bus is passive. The transmitter buffers revert to the PORT I/O function. The receive pins remain inputs. For the receiver, no error flags or Acknowledge signals are sent. The error counters are deactivated in this state. The filters are disabled. Receive Buffer 0 will receive any message transferred on the bus. This mode is useful to record all bus traffic as a bus monitor without influencing the data traffic.

## 23.6 Message Reception

This subsection describes CAN module message reception.

### 23.6.1 Receive Buffers

The CAN bus module has 3 receive buffers. However, one of the receive buffers is always committed to monitoring the bus for incoming messages. This buffer is called the message assembly buffer, MAB. So there are 2 receive buffers visible, RXB0 and RXB1, that can essentially instantaneously receive a complete message from the protocol engine. The CPU can be operating on one while the other is available for reception or holding a previously received message.

The MAB holds the destuffed bit stream from the bus line to allow parallel access to the whole data or remote frame for the acceptance match test and the parallel transfer of the frame to the receive buffers. The MAB will assemble all messages received. These messages will be transferred to the RXBn buffers only if the acceptance filter criterion are met. When a message is received, the RXnIF flag (CiINTF<0> or CiINRF<1>) will be set. This bit can only be set by the module when a message is received. The bit is cleared by the CPU when it has completed processing the message in the buffer. This bit provides a positive lockout to ensure that the CPU has finished with the message buffer. If the RXnIE bit (CiINTE<0> or CiINTE<1>) is set , an interrupt will be generated when a message is received.

There are 2 programmable acceptance filter masks associated with the receive buffers, one for each buffer.

When the message is received, the FILHIT bits (CiRX0CON<0> for Receive Buffer 0 and CiRX1CON<2:0> for Receive Buffer 1) indicate the acceptance criterion for the message. The number of the acceptance filter that enabled the reception will be indicated as well as a Status bit that indicates that the received message is a remote transfer request.

| **Note:** | In the case of Receive Buffer 0, a limited number of Acceptance Filters can be used to enable a reception. A single bit, FILHIT0 (CiRX0CON<0>) determines which of the 2 filters, RXF0 or RXF1, enabled the message reception. |
|---|---|

#### 23.6.1.1 Receive Buffer Priority

To provide flexibility, there are several acceptance filters corresponding to each receive buffer. There is also an implied priority to the receive buffers. RXB0 is the higher priority buffer and has 2 message acceptance filters associated with it. RXB1 is the lower priority buffer and has 4 acceptance filters associated with it. The lower number of possible acceptance filters makes the match on RXB0 more restrictive and implies the higher priority associated with that buffer. Additionally, if the RXB0 contains a valid message, and another valid message is received, the RXB0 can be setup such that it will not overrun and the new message for RXB0 will be placed into RXB1. Figure 23-9 shows a block diagram of the receive buffer, while Figure 23-10 shows a flow chart for a receive operation.

**Figure 23-9: The Receive Buffers**

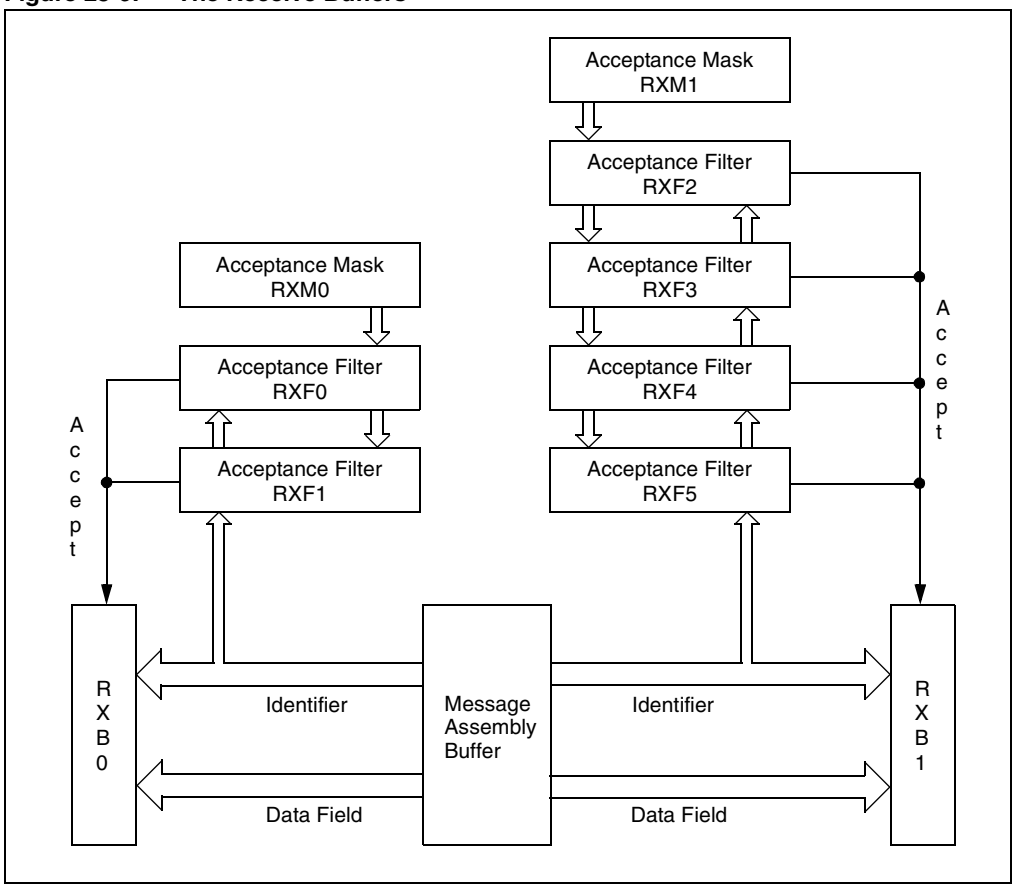
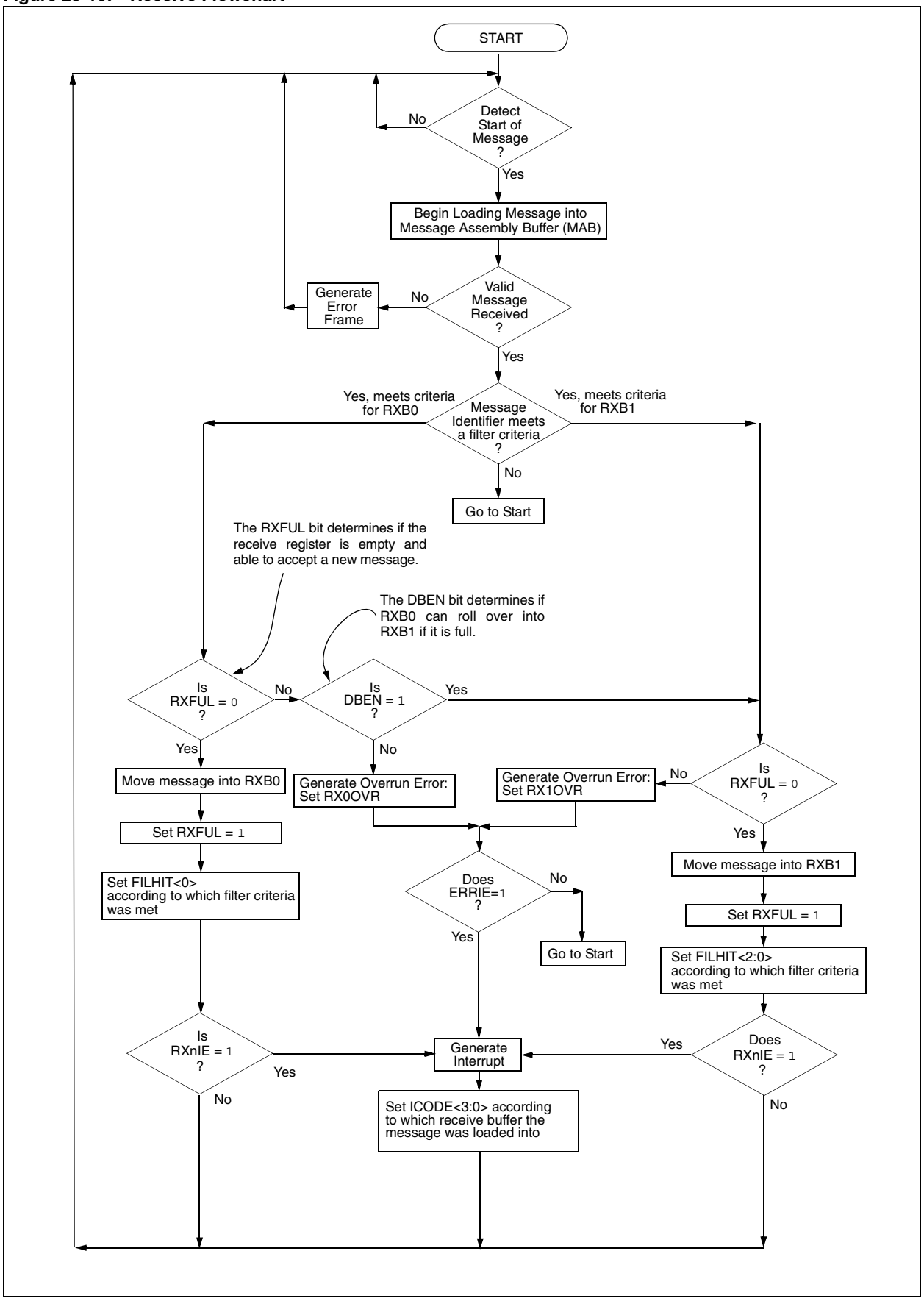


**Figure 23-10: Receive Flowchart**



### 23.6.2 Message Acceptance Filters

The message acceptance filters and masks are used to determine if a message in the message assembly buffer should be loaded into either of the receive buffers. Once a valid message has been received into the Message Assembly Buffer (MAB), the identifier fields of the message are compared to the filter values. If there is a match, that message will be loaded into the appropriate receive buffer. The filter masks are used to determine which bits in the identifiers are examined with the filters. A truthtable is shown in Table 23-3 that indicates how each bit in the identifier is compared to the masks and filters to determine if the message should be loaded into a receive buffer. The mask bit essentially determines which bits to apply the filter to. If any mask bit is set to a zero, then that bit will automatically be accepted regardless of the filter bit.

**Table 23-3: Filter/Mask Truth Table**

| Mask Bit n | Filter Bit n | Message Identifier bit | Accept or Reject bit n |
|---|---|---|---|
| 0 | x | x | Accept |
| 1 | 0 | 0 | Accept |
| 1 | 0 | 1 | Reject |
| 1 | 1 | 0 | Reject |
| 1 | 1 | 1 | Accept |

Legend: x = don't care

#### 23.6.2.1 Identifier Mode Selection

The EXIDE control bits (CiRXFnSID<0>) and the MIDE control bits (CiRXMnSID<0>) enable an acceptance filter for standard or extended identifiers. The acceptance filters look at incoming messages for the RXIDE bit to determine how to compare the identifiers. If the RXIDE bit is clear, the message is a standard frame. If the RXIDE bit is set, the message is an extended frame.

If the MIDE control bit for the filter is set, then the identifier type for the filter is determined by the EXIDE control bit for the filter. If the EXIDE control bit is cleared, then the filter will accept standard identifiers. If the EXIDE bit is set, then the filter will accept extended identifiers. Most CAN systems will use only standard identifiers or only extended identifiers.

If the MIDE control bit for the filter is cleared, the filter will accept both standard and extended identifiers if a match occurs with the filter bits. This mode can be used in CAN systems that support both standard and extended identifiers on the same bus.

#### 23.6.2.2 FILHIT Status Bits

As shown in the Receive Buffers Block Diagram, Figure 23-9, RXF0 and RXF1 filters with the RXM0 mask are associated with RXB0. The filters RXF2, RXF3, RXF4 and RXF5 and the mask RXM1 are associated with RXB1. When a filter matches and a message is loaded into the receive buffer, the number of the filter that enabled the message reception is indicated in the CiRXnCON register via the FILHIT bits. The CiRX0CON register contains one FILHIT Status bit to indicate whether the RXF0 or the RXF1 filter enabled the message reception. The CiRX1CON register contains the FILHIT<2:0> bits. They are coded as shown in Table 23-4.

**Table 23-4: Acceptance Filter**

| FILHIT<2:0> | Acceptance Filter | Comment |
|---|---|---|
| 000(1) | RXF0 | Only if DBEN = 1 |
| 001(1) | RXF1 | Only if DBEN = 1 |
| 010 | RXF2 | — |
| 011 | RXF3 | — |
| 100 | RXF4 | — |
| 101 | RXF5 | — |

**Note 1:** Is only valid if the DBEN bit is set.

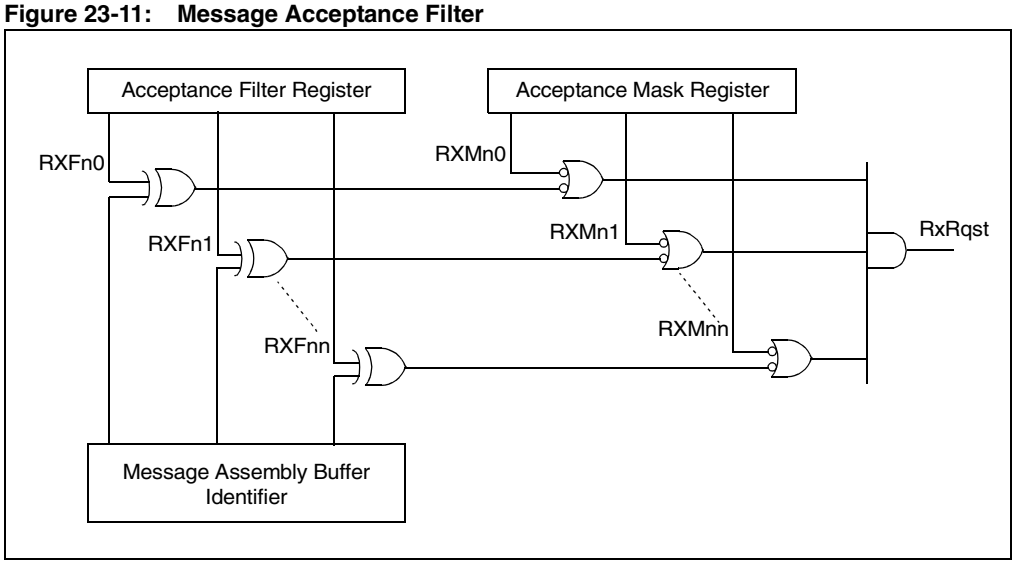
The DBEN bit (CiRX0CON<2>) allows the FILHIT bits to distinguish a hit on filter RXF0 and RXF1 in either RXB0 or overrun into RXB1.

111 = Acceptance Filter 1 (RXF1)
110 = Acceptance Filter 0 (RXF0)
001 = Acceptance Filter 1 (RXF1)
000 = Acceptance Filter 0 (RXF0)

If the DBEN bit is clear, there are 6 codes corresponding to the 6 filters. If the DBEN bit is set, there are 6 codes corresponding to the 6 filters plus 2 additional codes corresponding to RXF0 and RXF1 filters overrun to RXB1.

If more than 1 acceptance filter matches, the FILHIT bits will encode the lowest binary value of the filters that matched. In other words, if filter 2 and filter 4 match, FILHIT will code the value for 2. This essentially prioritizes the acceptance filters with lower numbers having priority. Figure 23-11 shows a block diagram of the message acceptance filters.

**Figure 23-11: Message Acceptance Filter**



### 23.6.3 Receiver Overrun

An overrun condition occurs when the Message Assembly Buffer (MAB) has assembled a valid received message, the message is accepted through the acceptance filters, and when the receive buffer associated with the filter has not been designated as clear of the previous message.

The overrun error flag, RXnOVR (CiINTF<15> or CiINTF<14>) and the ERRIF bit (CiINTF<5>) will be set and the message in the MAB will be discarded. While in the overrun situation, the module will stay synchronized with the CAN bus and is able to transmit messages, but it will discard all incoming messages destined for the overflowed buffer.

If the DBEN bit is clear, RXB1 and RXB0 operate independently. When this is the case, a message intended for RXB0 will not be diverted into RXB1 if RXB0 contains an unread message and the RX0OVR bit will be set.

If the DBEN bit is set, the overrun for RXB0 is handled differently. If a valid message is received for RXB0 and RXFUL = 1 (CiRX0CON<7>) indicating that RXB0 is full, and RXFUL = 0 (CiRX1CON<7>) indicating that RXB1 is empty, the message for RXB0 will be loaded into RXB1. An overrun error will not be generated for RXB0. If a valid message is received for RXB0 and RXFUL = 1, and RXFUL = 1 indicating that both RXB0 and RXB1 are full, the message will be lost and an overrun will be indicated for RXB1.

If the DBEN bit is clear, there are six codes corresponding to the six filters. If the DBEN bit is set, there are six codes corresponding to the six filters plus two additional codes corresponding to RXF0 and RXF1 filters overrun to RXB1. These codes are given in Table 23-5.

**Table 23-5: Buffer Reception and Overflow Truth Table**

| Message Matches Filter 0 or 1 | Message Matches Filter 2,3,4,5 | RXFUL0 Bit | RXFUL1 Bit | DBEN Bit | Action | Results |
|---|---|---|---|---|---|---|
| 0 | 0 | X | X | X | None | No message received |
| 0 | 1 | X | 0 | X | MAB → RXB1 | Message for RXB1, RXB1 available |
| 0 | 1 | X | 1 | X | MAB discarded RX1OVR = 1 | Message for RXB1, RXB1 full |
| 1 | 0 | 0 | X | X | MAB → RXB0 | Message for RXB0, RXB0 available |
| 1 | 0 | 1 | X | 0 | MAB discarded RX0OVR = 1 | Message for RXB0, RXB0 full, DBEN not enabled |
| 1 | 0 | 1 | 0 | 1 | MAB → RXB1 | Message for RXB0, RXB0 full, DBEN enabled, RXB1 available |
| 1 | 0 | 1 | 1 | 1 | MAB discarded RX1OVR = 1 | Message for RXB0, RXB0 full, DBEN enabled, RXB1 full |
| 1 | 1 | 0 | X | X | MAB → RXB0 | Message for RXB0 and RXB1, RXB0 available |
| 1 | 1 | 1 | X | 0 | MAB discarded RX0OVR = 1 | Message for RXB0 and RXB1, RXB0 full, DBEN not enabled |
| 0 | 0 | X | X | X | None | No message received |
| 0 | 1 | X | 0 | X | MAB → RXB1 | Message for RXB1, RXB1 available |

Legend: X = Don't care

### 23.6.4 Effects of a RESET

Upon any RESET the CAN module has to be initialized. All registers are set according to the RESET values. The content of a received message is lost. The initialization is discussed in **Section 23.5.5 "Configuration Mode"**.

### 23.6.5 Receive Errors

The CAN module will detect the following receive errors:

- Cyclic Redundancy Check (CRC) Error
- Bit Stuffing Error
- Invalid message receive error

These receive errors do not generate an interrupt. However, the receive error counter is incremented by one in case one of these errors occur. The RXWAR bit (CiINTF<9>) indicates that the Receive Error Counter has reached the CPU warning limit of 96 and an interrupt is generated.

#### 23.6.5.1 Cyclic Redundancy Check (CRC) Error

With the Cyclic Redundancy Check, the transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the data field. This CRC sequence is transmitted in the CRC Field. The receiving node also calculates the CRC sequence using the same formula and performs a comparison to the received sequence. If a mismatch is detected, a CRC error has occurred and an Error Frame is generated. The message is repeated. The receive error interrupt counter is incremented by one. An Interrupt will only be generated if the error counter passes a threshold value.

#### 23.6.5.2 Bit Stuffing Error

If, between the Start -Of-Frame and the CRC Delimiter, 6 consecutive bits with the same polarity are detected, the bit-stuffing rule has been violated. A bit-stuffing error occurs and an error frame is generated. The message is repeated. No interrupt will be generated upon this event.

#### 23.6.5.3 Invalid Message Received Error

If any type of error occurs during reception of a message, an error will be indicated by the IVRIF bit (CiINTF<7>). This bit can be used (optionally with an interrupt) for autobaud detection with the device in Listen Only mode. This error is not an indicator that any action needs to be taken, but it does indicate that an error has occurred on the CAN bus.

#### 23.6.5.4 Rules for Modifying the Receive Error Counter

The Receive Error Counter is modified according to the following rules:

- When the receiver detects an error, the Receive Error Counter is incremented by 1, except when the detected error was a bit error during the transmission of an active error flag or an overload flag.
- When the receiver detects a "dominant" bit as the first bit after sending an error flag, the Receive Error Counter will be incremented by 8.
- If a receiver detects a bit error while sending an active error flag or an overload flag, the Receive Error Counter is incremented by 8.
- Any node tolerates up to 7 consecutive "dominant" bits after sending an active error flag, passive error flag or an overload flag. After detecting the 14th consecutive "dominant" bit (in case of an Active error flag or an Overload flag) or after detecting the 8th consecutive "dominant" bit following a passive error flag, and after each sequence of eight additional consecutive "dominant" bits, every transmitter increases its Transmission Error Counter and every receiver increases its Receive Error Counter by 8.
- After a successful reception of a message (reception without error up to the ACK slot and the successful sending of the ACK bit), the Receive Error Counter is decreased by one, if the Receive Error Counter was between 1 and 127. If the Receive Error Counter was '0', it will stay '0'. If the Receive Error Counter was greater than 127, it will change to a value between 119 and 127.

### 23.6.6 Receive Interrupts

Several Interrupts are linked to the message reception. The receive interrupts can be broken up into two separate groups:

- Receive Error Interrupts
- Receive interrupts

#### 23.6.6.1 Receive Interrupt

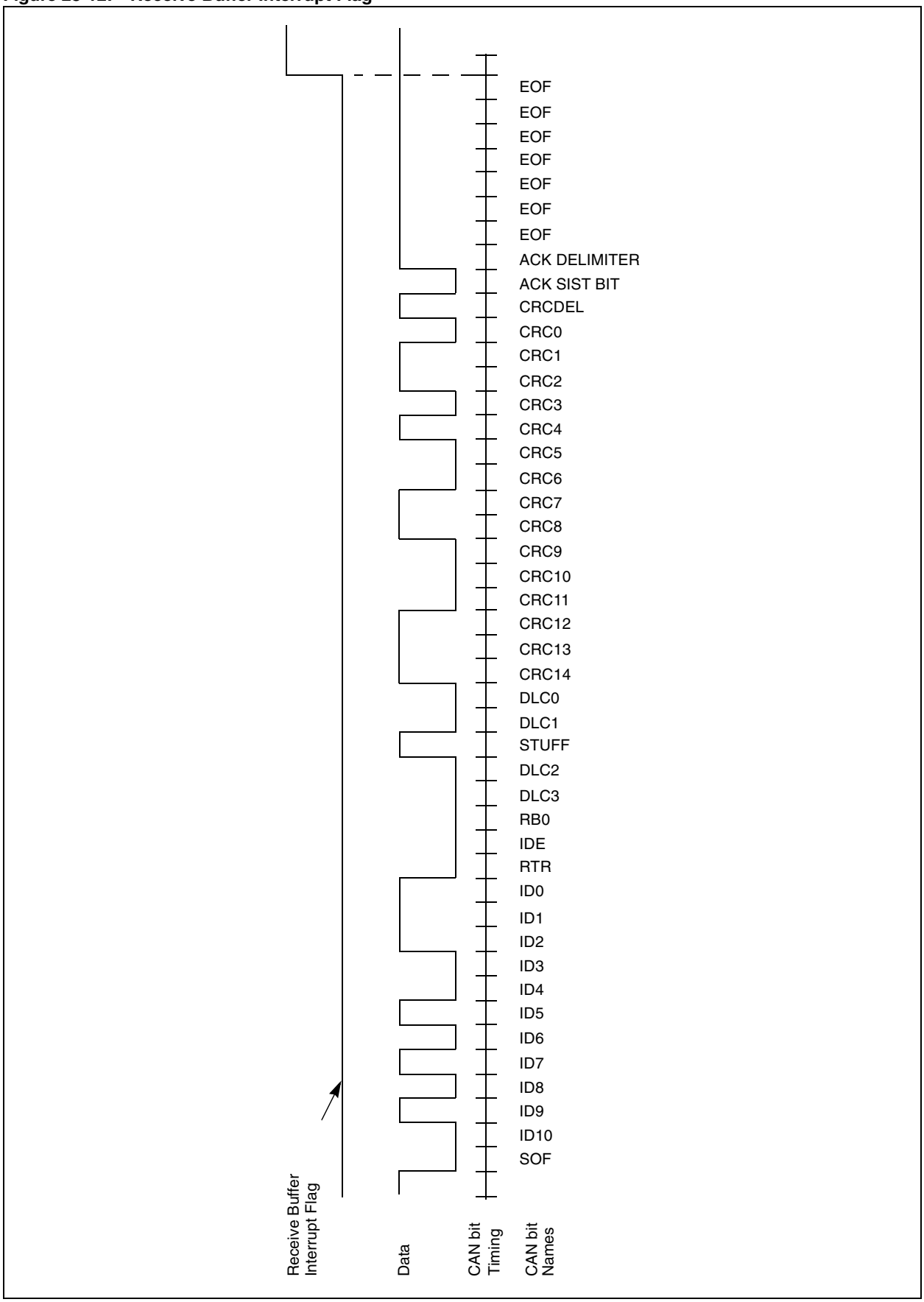
A message has been successfully received and loaded into one of the receive buffers. This interrupt is activated immediately after receiving the End-Of-Frame (EOF) field. Reading the RXnIF flag will indicate which receive buffer caused the interrupt. Figure 23-12 depicts when the receive buffer interrupt flag RXnIF will be set.

#### 23.6.6.2 Wake-up Interrupt

The Wake-up interrupt sequences are described in **Section 23.13.1 "Operation in SLEEP Mode"**.

**Figure 23-12: Receive Buffer Interrupt Flag**



#### 23.6.6.3 Receive Error Interrupts

A receive error interrupt will be indicated by the ERRIF bit (CiINTF<5>). This bit shows that an error condition occurred. The source of the error can be determined by checking the bits in the CAN Interrupt Status Register CiINTF. The bits in this register are related to receive and transmit errors. The following subsequences will show which flags are linked to the receive errors.

##### 23.6.6.3.1 Invalid Message Received Interrupt

If any type of error occurred during reception of the last message, an error will be indicated by the IVRIF bit (CiINTF<7>). The specific error that occurred is unknown. This bit can be used (optionally with an interrupt) for autobaud detection with the device in Listen Only mode. This error is not an indicator that any action needs to be taken, but an indicator that an error has occurred on the CAN bus.

##### 23.6.6.3.2 Receiver Overrun Interrupt

The RXnOVR bit (CiINTF<15>, CiINTF<14>) indicates that an overrun condition occurred for the receive buffer. An overrun condition occurs when the Message Assembly Buffer (MAB) has assembled a valid received message, the message is accepted through the acceptance filters, however, the receive buffer associated with the filter is not clear of the previous message. The overflow error interrupt will be set and the message is discarded. While in the overrun situation, the module will stay synchronized with the CAN bus and is able to transmit and receive messages.

#### 23.6.6.4 Receiver Warning Interrupt

The RXWAR bit (CiINTF<8>) indicates that the Receive Error Counter has reached the CPU warning limit of 96. When RXWAR transitions from a ‘0’ to a ‘1’, it will cause the Error Interrupt Flag ERRIF to become set. This bit cannot be manually cleared, as it should remain an indicator that the Receive Error Counter has reached the CPU warning limit of 96. The RXWAR bit will become clear automatically if the Receive Error Counter becomes less than or equal to 95. The ERRIF bit can be manually cleared allowing the interrupt service routine to be exited without affecting the RXWAR bit.

#### 23.6.6.5 Receiver Error Passive

The RXEP bit (CiINTF<11>) indicates that the Receive Error Counter has exceeded the Error Passive limit of 127 and the module has gone to Error Passive state. When the RXEP bit transitions from a ‘0’ to a ‘1’, it will cause the error interrupt flag to become set. The RXEP bit cannot be manually cleared, as it should remain an indicator that the bus is in Error State Passive. The RXEP bit will become clear automatically if the Receive Error Counter becomes less than or equal to 127. The ERRIF bit can be manually cleared allowing the interrupt service routine to be exited without affecting the RXEP bit.

## 23.7 Transmission

This subsection describes how the CAN module is used to transmit CAN messages.

### 23.7.1 Real Time Communication and Transmit Message Buffering

For an application to effectively transmit messages in real-time, the CAN nodes must be able to dominate and hold the bus, assuming that nodes messages are of a high enough priority to win arbitration on the bus. If a node only has 1 transmission buffer, it must transmit a message, then release the bus while the CPU reloads the buffer. If a node has two transmission buffers, one buffer could be transmitting while the second buffer is being reloaded. However, the CPU would need to maintain tight tracking of the bus activity to ensure that the second buffer is reloaded before the first message completes.

Typical applications require three transmit message buffers. With three buffers, one buffer can be transmitting, the second buffer can be ready to transmit as soon as the first is complete, and the third can be reloaded by the CPU. This eases the burden of the software to maintain synchronization with the bus (see Figure 23-13).

Additionally, the three buffers allow some degree of prioritizing of the outgoing messages. For example, the application software may have a message enqueued in the second buffer while it is working on the third buffer. The application may require that the message going into the third buffer is of higher importance than the one already enqueued. If only 2 buffers are available, the enqueued message would have to be deleted and replaced with the third. The process of deleting the message may mean losing control of the bus. With 3 buffers, both the second and the third message can be enqueued, and the module can be instructed that the third message is higher priority than the second. The third message will be the next one sent followed by the second.

### 23.7.2 Transmit Message Buffers

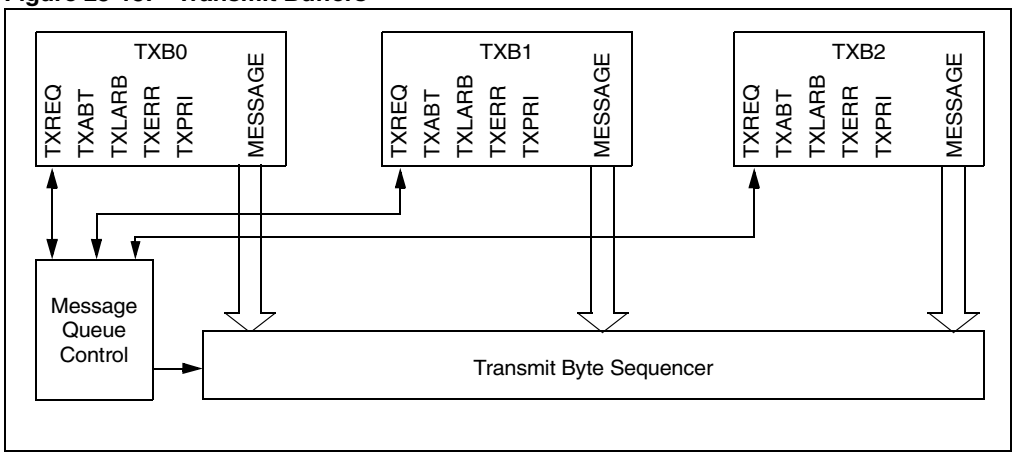
The CAN module has three Transmit Buffers. Each of the three buffers occupies 14 bytes of data. Eight of the bytes are the maximum 8 bytes of the transmitted message. Five bytes hold the standard and extended identifiers and other message arbitration information.

The last byte is a control byte associated with each message. The information in this byte determines the conditions under which the message will be transmitted and indicates status of the transmission of the message.

The TXnIF bit (CiINTF<2>, CiINTF<3> or CiINTF<4>) will be set and the TXREQ bit (CiTXnCON<3>) will be clear, indicating that the message buffer has completed a transmission. The CPU will then load the message buffer with the contents of the message to be sent. At a minimum, the standard identifier register CiTXnSID must be loaded. If data bytes are present in the message, the TXBnDm registers are loaded. If the message is to use extended identifiers, the CiTXnEID register and the EID<5:0> bits (CiTXnDLC<15:10>) are loaded and the TXIDE bit is set (CiTXnSID<0>).

Prior to sending the message, the user must initialize the TXnIE bit (CiINTE<2>, CiINTE<3> or CiINTE<4>) to enable or disable an interrupt when the message is sent. The user must also initialize the transmit priority. Figure 23-13 shows a block diagram of the Transmit Buffers.

**Figure 23-13: Transmit Buffers**




### 23.7.3 Transmit Message Priority

Transmit priority is a prioritization within each node of the pending transmittable messages. Prior to sending the SOF (Start-Of-Frame), the priorities of all buffers ready for transmission are compared. The Transmit Buffer with the highest priority will be sent first. For example, if Transmit Buffer 0 has a higher priority setting than Transmit Buffer 1, Buffer 0 will be sent first. If two buffers have the same priority setting, the buffer with the highest address will be sent. For example, if Transmit Buffer 1 has the same priority setting as Transmit Buffer 0, Buffer 1 will be sent first. There are 4 levels of transmit priority. If TXPRI<1:0> (CiTXnCON<1:0>) for a particular message buffer is set to '11', that buffer has the highest priority. If TXPRI<1:0> for a particular message buffer is set to '10' or '01', that buffer has an intermediate priority. If TXPRI<1:0> for a particular message buffer is '00', that buffer has the lowest priority.

### 23.7.4 Message Transmission

To initiate transmitting the message, the TXREQ bit (CiTXnCON<3>) must be set. The CAN bus module resolves any timing conflicts between setting of the TXREQ bit and the SOF time, ensuring that if the priority was changed, it is resolved correctly before SOF. When TXREQ is set the TXABT (CiTXnCON<6>), TXLARB (CiTXnCON<5>) and TXERR (CiTXnCON<4>) flag bits will be cleared by the module.

Setting TXREQ bit does not actually start a message transmission, it flags a message buffer as enqueued for transmission. Transmission will start when the module detects an available bus for SOF. The module will then begin transmission on the message which has been determined to have the highest priority.

If the transmission completes successfully on the first try, the TXREQ bit will clear and an interrupt will be generated if the TXnIE bit (CiINTE<2>, CiINTE<3>, CiINTE<4>) is set.

If the message fails to transmit, other condition flags will be set and the TXREQ bit will remain set indicating that the message is still pending for transmission. If the message tried to transmit but encountered an error condition, the TXERR bit (CiTXnCON<4>) will be set. In this case, the error condition can also cause an interrupt. If the message tried to transmit but lost arbitration, the TXLARB bit (CiTXnCON<5>) will be set. In this case, no interrupt is available to signal the loss of arbitration.

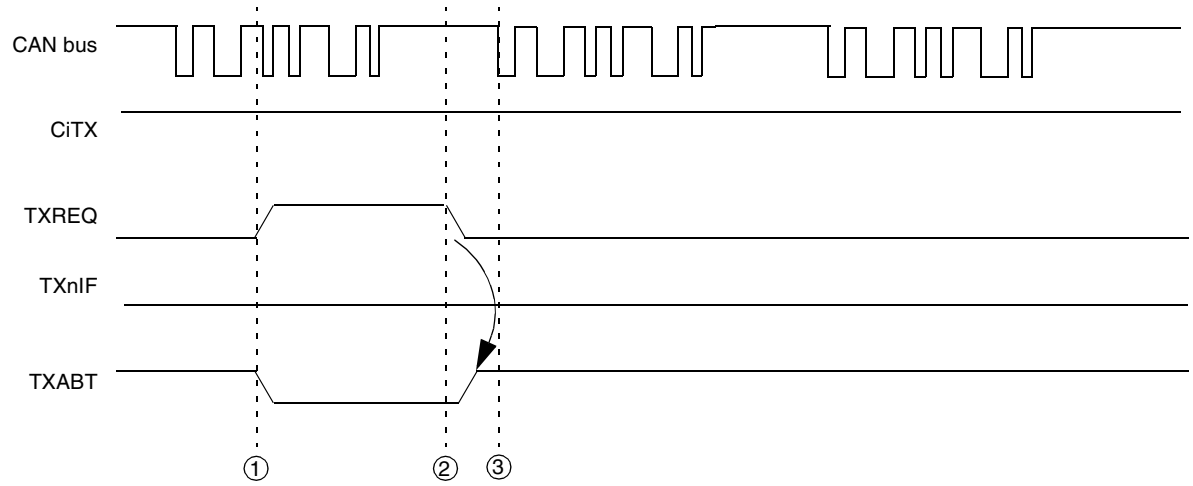
 

### 23.7.5 Transmit Message Aborting

The system can abort a message by clearing the TXREQ bit associated with each message buffer. Setting the ABAT bit (CiCTRL<12>) will request an abort of all pending messages (see Figure 23-15). A queued message is aborted by clearing the TXREQ bit. Aborting a queued message is illustrated in Figure 23-14. If the message has not yet started transmission, or if the message started but is interrupted by loss of arbitration or an error; the abort will be processed. The abort is indicated when the module sets the TXABT bit (CiTXnCON<6>), and the TXnIF flag is not set.

If the message has started to transmit, it will attempt to transmit the current message fully (see Figure 23-16). If the current message is transmitted fully, and is not lost to arbitration or an error, the TXABT bit will not be set, because the message was transmitted successfully. Likewise, if a message is being transmitted during an abort request, and the message is lost to arbitration (see Figure 23-17) or an error, the message will not be re-transmitted, and the TXABT bit will be set, indicating that the message was successfully aborted.

**Figure 23-14: Abort Queued Message**

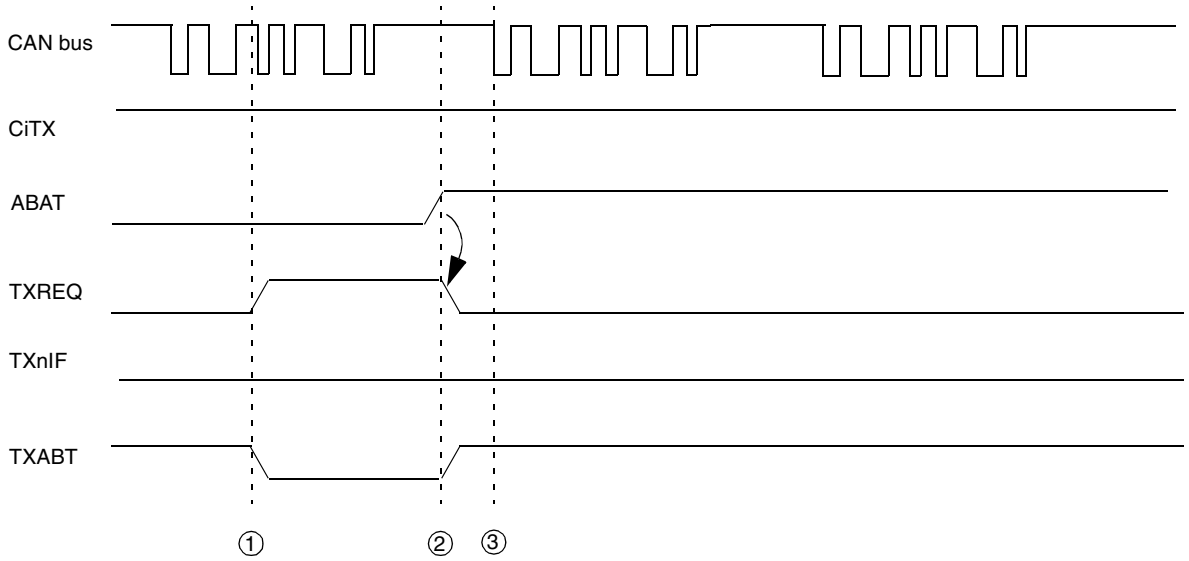



① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

② - Processor clears TXREQ while module looking for 11 recessive bits.
Module aborts pending transmission, sets TXABT bit in 2 clocks.

③ - Another module takes the available transmit slot.

**Figure 23-15: Abort All Messages**

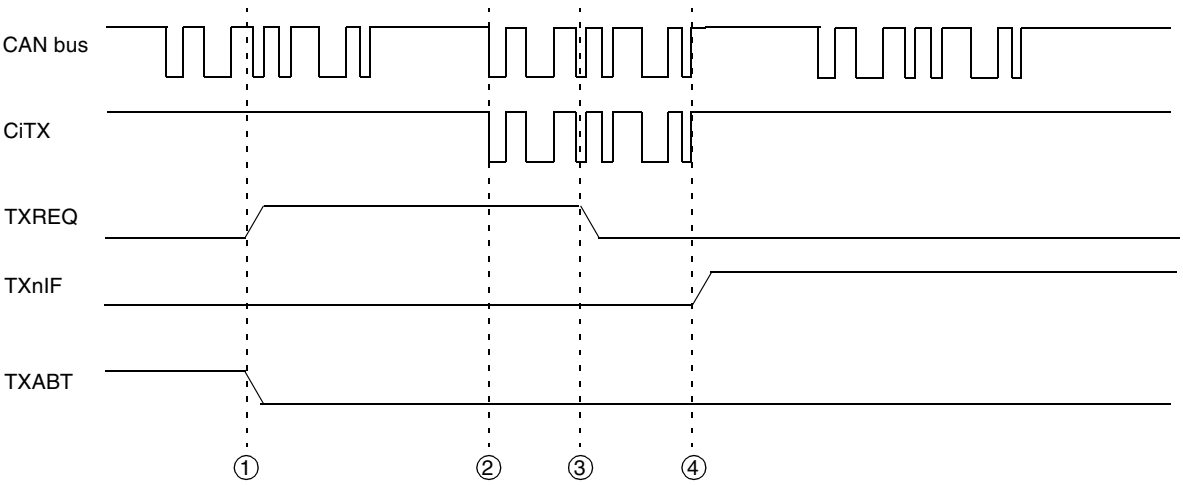



① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

② - Processor sets ABAT while module looking for 11 recessive bits. Module clears TXREQ bits. Module aborts pending transmission, sets TXABT bit.

③ - Another module takes the available transmit slot.

**Figure 23-16: Failed Abort During Transmission**




① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

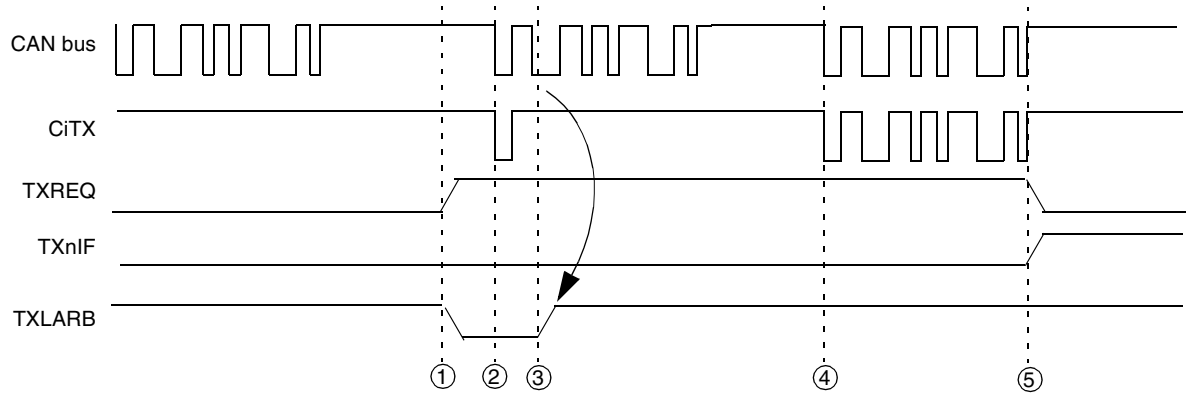
② - Module detects 11 recessive bits. Module begins transmission of queued message.

③ - Processor clears TXREQ requesting message abort. Abort cannot be acknowledged.

④ - At successful completion of transmission, TXREQ bit remains clear and TXnIF bit set. TXABT remains clear.

**Figure 23-17: Loss of Arbitration During Transmission**




① - Processor sets TXREQ while module inactive. TXLARB bit cleared.

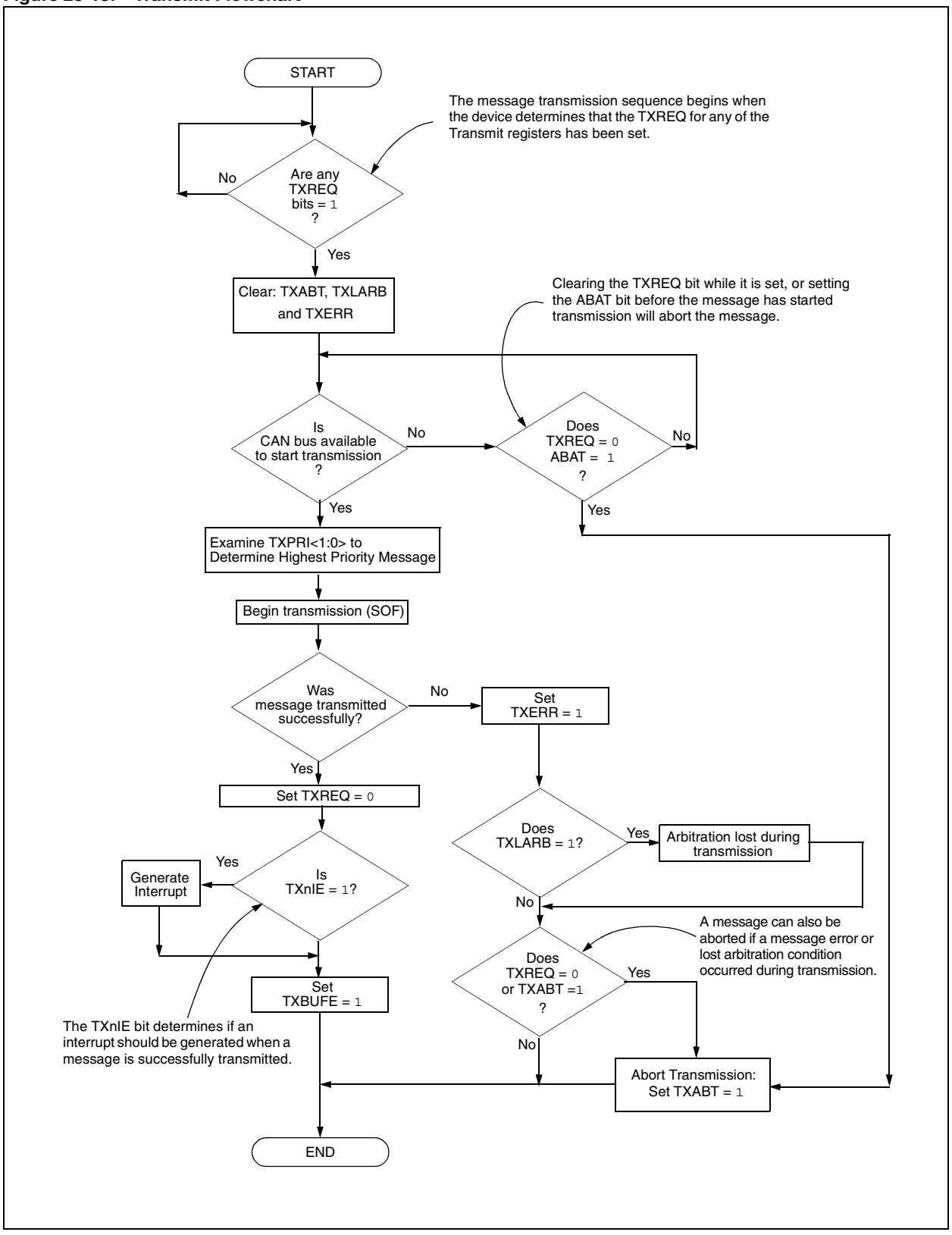
② - Module in inactive state. Module begins transmission of queued message.

③ - Message loses arbitration. Module releases bus and sets TXLARB bit.

④ - Module waits for 11 recessive bits before re-trying transmission of queued message.

⑤ - At successful completion of transmission, TXREQ bit cleared and TXnIF bit set.

**Figure 23-18: Transmit Flowchart**



### 23.7.6 Transmit Boundary Conditions

The module handles transmit commands which are not necessarily synchronized to the CAN bus message framing time.

#### 23.7.6.1 Clearing TXREQ bit as a Message Starts

The TXREQ bit can be cleared just when a message is starting transmission, with the intent of aborting the message. If the message is not being transmitted, the TXABT bit will be set, indicating that the Abort was successfully processed.

When the user clears the TXREQ bit and the TXABT bit is not set two cycles later, the message has already begun transmission.

If the message is being transmitted, the abort is not immediately processed, at some point later, the TXnIF interrupt flag or the TXABT bit is set. If transmission has begun the message will only be aborted if either an error or a loss of arbitration occurs.

#### 23.7.6.2 Setting TXABT bit as a Message Starts

Setting the ABAT bit will abort all pending Transmit Buffers and has the function of clearing all of the TXREQ bits for all buffers. The boundary conditions are the same as clearing the TXREQ bit.

#### 23.7.6.3 Clearing TXREQ bit as a Message Completes

The TXREQ bit can be cleared when a message is just about to successfully complete transmission. Even if the TXREQ bit is cleared by the Data bus a short time before it will be cleared by the successful transmission of the message, the TXnIF flag will still be set due to the successful transmission.

#### 23.7.6.4 Setting TXABT bit as a Message Completes

The boundary conditions are the same as clearing the TXREQ bit.

#### 23.7.6.5 Clearing TXREQ bit as a Message Loses Transmission

The TXREQ bit can be cleared when a message is just about to be lost to arbitration or an error.

If the TXREQ signal falls before the loss of arbitration signal or error signal, the result will be like clearing TXREQ during transmission. When the arbitration is lost or the error is set, the TXABT bit will be set, as it will see that an error has occurred while transmitting, and that the TXREQ bit was not set.

If the TXREQ bit falls after the arbitration signal has entered the block, the result will be like clearing TXREQ during an inactive transmit time. The TXABT bit will be set.

#### 23.7.6.6 Setting TXABT bit as a Message Loses Transmission

The boundary conditions are the same as clearing the TXREQ bit.

### 23.7.7 Effects of a RESET

Upon any RESET the CAN module has to be initialized. All registers are set according to the reset values. The content of a transmitted message is lost. The initialization is discussed in **Section 23.5.5 “Configuration Mode”**.

### 23.7.8 Transmission Errors

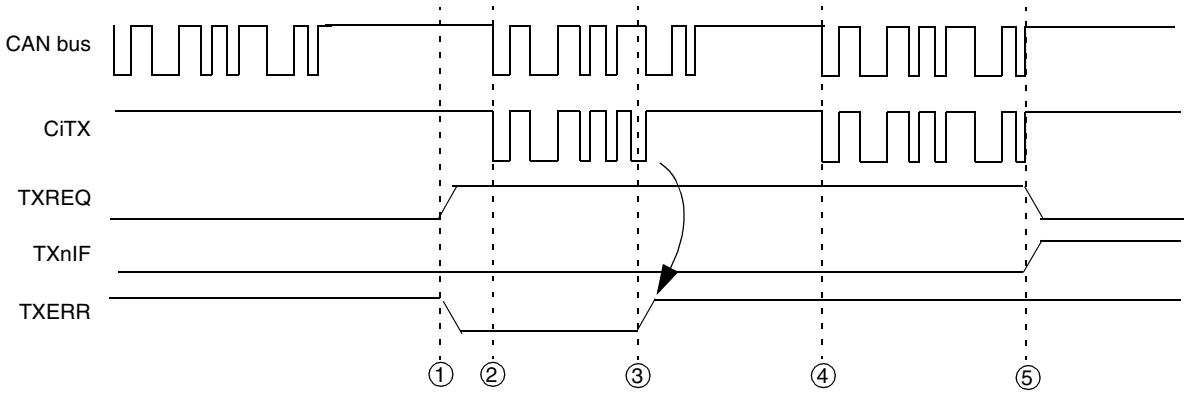
The CAN module will detect the following transmission errors:

- Acknowledge Error
- Form Error
- Bit Error

These transmission errors will not necessarily generate an interrupt but are indicated by the transmission error counter. However, each of these errors will cause the transmission error counter to be incremented by one. Once the value of the error counter exceeds the value of 96, the ERRIF (CiINTF<5>) and the TXWAR bit (CiINTF<10>) are set. Once the value of the error counter exceeds the value of 96 an interrupt is generated and the TXWAR bit in the error flag register is set.

A transmission error example is illustrated in Figure 23-19.

**Figure 23-19: Error During Transmission**




① - Processor sets TXREQ while module inactive. TXERR bit is cleared.
② - Module in inactive state. Module begins transmission of queued message.
③ - Module detects error during transmission, releases bus and sets TXERR bit.
④ - Module waits for 11 recessive bits before re-trying transmission of queued message.
⑤ - At successful completion of transmission, TXREQ bit cleared and TXnIF bit set.

#### 23.7.8.1 Acknowledge Error

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

#### 23.7.8.2 Form Error

If a transmitter detects a dominant bit in one of the four segments including End-Of-Frame, Interframe Space, Acknowledge Delimiter or CRC Delimiter; then a form error has occurred and an error frame is generated. The message is repeated.

#### 23.7.8.3 Bit Error

A bit error occurs if a transmitter sends a dominant bit and detects a recessive bit. In the case where the transmitter sends a recessive bit and a dominant bit is detected during the Arbitration field and the Acknowledge Slot, no bit error is generated because normal arbitration is occurring.

#### 23.7.8.4 Rules for Modifying the Transmit Error Counter

The Transmit Error Counter is modified according to the following rules:

- When the transmitter sends an error flag the Transmit Error Counter is increased by 8 with the following exceptions. In these two exceptions, the Transmit Error Counter is not changed.
  - If the transmitter is “error passive” and detects an acknowledgment error because of not detecting a “dominant” ACK, and does not detect a “dominant” bit while sending a passive error flag.
  - If the transmitter sends an error flag because of a bit-stuffing error occurred during arbitration whereby the Stuffbit is located before the RTR bit, and should have been “recessive”, and has been sent as “recessive” but monitored as “dominant”.
- If a transmitter detects a bit error while sending an active error flag or an overload flag the Transmit Error Counter is increased by 8.
- Any Node tolerates up to 7 consecutive “dominant” bits after sending an active error flag, passive error flag or an overload flag. After detecting the 14th consecutive “dominant” bit (in case of an active error flag or an overload flag) or after detecting the 8th consecutive “dominant” following a passive error flag, and after each sequence of eight additional consecutive “dominant” bits, every transmitter increases its Transmission Error Counter and every receiver increases its Receive Error Counter by 8.
- After the successful transmission of a message (getting an acknowledge and no error until End-Of-Frame is finished) the Transmit Error Counter is decreased by one unless it was already 0.

### 23.7.9 Transmission Interrupts

There are several interrupts linked to the message transmission. The transmission interrupts can be broken up into two groups:

- Transmission interrupts
- Transmission error interrupts

#### 23.7.9.1 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. Reading the TXnIF flags in the CiINTF register will indicate which transmit buffer is available and caused the interrupt.

#### 23.7.9.2 Transmission Error Interrupts

A transmission error interrupt will be indicated by the ERRIF flag. This flag shows that an error condition occurred. The source of the error can be determined by checking the error flags in the CAN Interrupt Status register CiINTF. The flags in this register are related to receive and transmit errors.

The TXWAR bit (CiINTF<10>) indicates that the Transmit Error Counter has reached the CPU warning limit of 96. When this bit transitions from a '0' to a '1', it will cause the error interrupt flag to become set. The TXWAR bit cannot be manually cleared, as it should remain as an indicator that the Transmit Error Counter has reached the CPU warning limit of 96. The TXWAR bit will become clear automatically if the Transmit Error Counter becomes less than or equal to 95. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXWAR bit.

The TXEP bit (CiINTF<12>) indicates that the Transmit Error Counter has exceeded the Error Passive limit of 127 and the module has gone to Error Passive state. When this bit transitions froma '0' to a '1', it will cause the error interrupt flag to become set. The TXEP bit cannot be manually cleared, as it should remain as an indicator that the bus is in Error Passive state. The TXEP bit will become clear automatically if the Transmit Error Counter becomes less than or equal to 127. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXEP bit.

The TXBO bit (CiINTF<13>) indicates that the Transmit Error Counter has exceeded 255 and the module has gone to bus off state. When this bit transitions from a '0' to a '1', it will cause the error interrupt flag to become set. The TXBO bit cannot be manually cleared, as it should remain as an indicator that the bus is off. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXBO bit.

### 23.8 Error Detection

The CAN protocol provides sophisticated error detection mechanisms. The following errors can be detected. These errors are either receive or transmit errors.

Receive errors are:

- Cyclic Redundancy Check (CRC) Error (see **Section 23.6.5.1 "Cyclic Redundancy Check (CRC) Error"**)
- Bit Stuffing Bit Error (see **Section 23.6.5.2 "Bit Stuffing Error"**)
- Invalid Message Received Error (see **Section 23.6.5.3 "Invalid Message Received Error"**)

The transmit errors are:

- Acknowledge Error (see **Section 23.7.8.1 "Acknowledge Error"**)
- Form Error (see **Section 23.7.8.2 "Form Error"**)
- Bit Error (see **Section 23.7.8.3 "Bit Error"**)

### 23.8.1 Error States

Detected errors are made public to all other nodes via error frames. The transmission of the erroneous message is aborted and the frame is repeated as soon as possible. Furthermore, each CAN node is in one of the three error states "error active", "error passive" or "bus off" according to the value of the internal error counters. The error active state is the usual state where the bus node can transmit messages and active error frames (made of dominant bits) without any restrictions. In the error passive state, messages and passive error frames (made of recessive bits) may be transmitted. The bus off state makes it temporarily impossible for the station to participate in the bus communication. During this state, messages can neither be received nor transmitted.

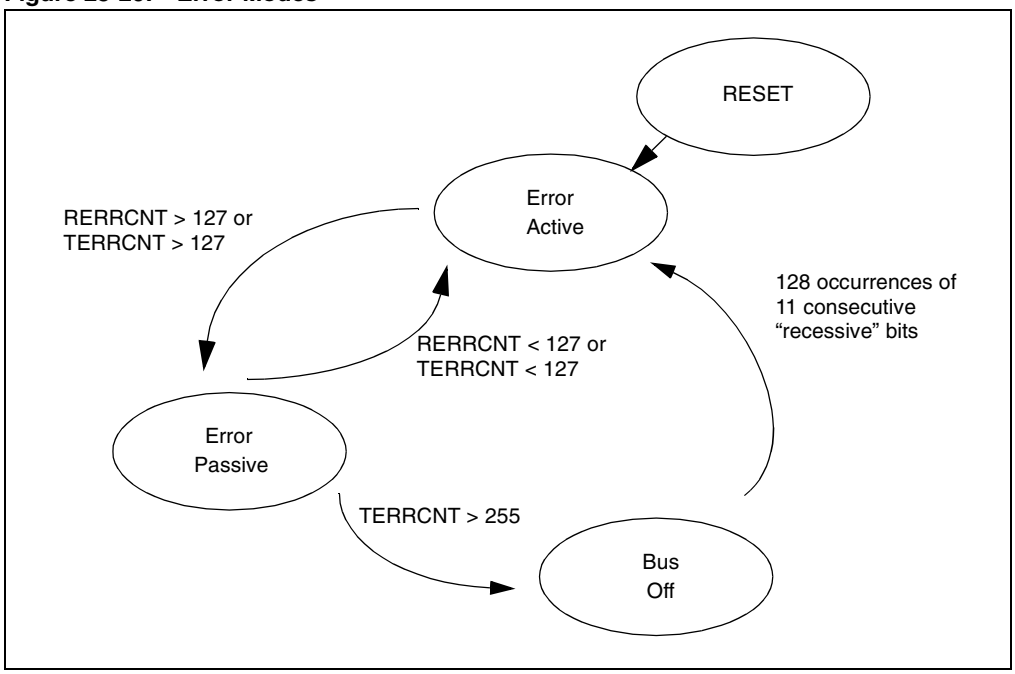
 

### 23.8.2 Error Modes and Error Counters

The CAN controller contains the two error counters Receive Error Counter (RERRCNT) and Transmit Error Counter (TERRCNT). The values of both counters can be read by the CPU from the Error Count Register CiEC. These counters are incremented or decremented according to the CAN bus specification.

The CAN controller is error active if both error counters are below the error passive limit of 128. It is error passive if at least one of the error counters equals or exceeds 128. It goes bus off if the Transmit Error Counter equals or exceeds the bus off limit of 256. The device remains in this state, until the bus off recovery sequence is finished, which is 128 consecutive 11 recessive bit times. Additionally, there is a error state warning flag bit, EWARN (CiINTF<8>), which is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWARN is reset if both error counters are less than the error warning limit.

**Figure 23-20: Error Modes**




### 23.8.3 Error Flag Register

The values in the error flag register indicate which error(s) caused the error interrupt flag. The RXnOVR error flags (CiINTF<15> and CiINTF<14>) have a different function than the other error flag bits in this register. The RXnOVR bits must be cleared in order to clear the ERRIF interrupt flag. The other error flag bits in this register will cause the ERRIF interrupt flag to become set as the value of the Transmit and Receive Error Counters crosses a specific threshold. Clearing the ERRIF interrupt flag in these cases will allow the interrupt service routine to be exited without recursive interrupt occurring. It may be desirable to disable specific interrupts after they have occurred once to stop the device from interrupting repeatedly as the Error Counter moves up and down in the vicinity of a threshold value.

### 23.9 CAN Baud Rate

All nodes on any particular CAN bus must have the same nominal bit rate. The CAN bus uses NRZ coding which does not encode a clock. Therefore the receivers independent clock must be recovered by the receiving nodes and synchronized to the transmitters clock.

In order to set the baud rate the following bits have to be initialized:

- Synchronization Jump Width (see Section **Section 23.9.6.2 "Re-synchronization"**)
- Baud Rate Prescaler (see Section **Section 23.9.2 "Prescaler Setting"**)
- Phase Segments (see Section **Section 23.9.4 "Phase Segments"**)
- Length Determination of Phase Segment 2 (see Section **Section 23.9.4 "Phase Segments"**)
- Sample Point (see Section **Section 23.9.5 "Sample Point"**)
- Propagation Segment Bits (see Section **Section 23.9.3 "Propagation Segment"**)

### 23.9.1 Bit Timing

As oscillators and transmission time may vary from node to node, the receiver must have some type of PLL synchronized to data transmission edges to synchronize and maintain the receiver clock. Since the data is NRZ coded, it is necessary to include bit-stuffing to ensure that an edge occurs at least every 6 bit times, to maintain the Digital Phase Lock Loop (DPLL) synchronization.

Bus timing functions executed within the bit time frame, such as synchronization to the local oscillator, network transmission delay compensation, and sample point positioning, are defined by the programmable bit timing logic of the DPLL.

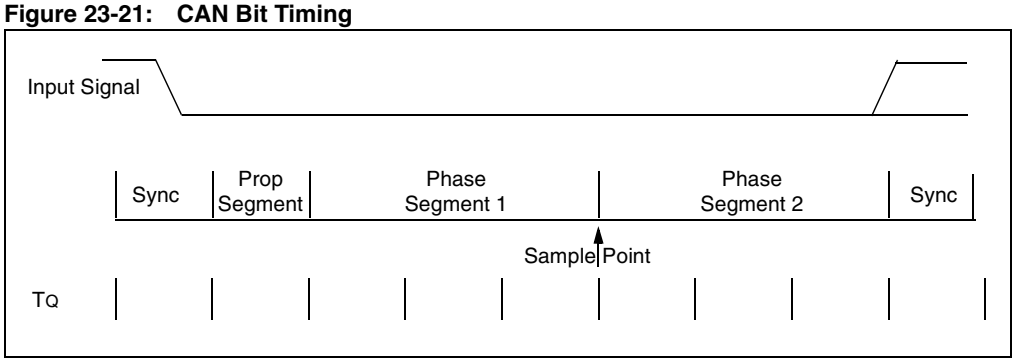
All controllers on the CAN bus must have the same baud rate and bit length. However, different controllers are not required to have the same master oscillator clock. At different clock frequencies of the individual controllers, the baud rate has to be adjusted by adjusting the number of time quanta in each segment.

The nominal bit time can be thought of as being divided into separate non-overlapping time segments. These segments are shown in Figure 23-21.

- Synchronization segment (Sync Seg)
- Propagation time segment (Prop Seg)
- Phase buffer segment 1 (Phase1 Seg)
- Phase buffer segment 2 (Phase2 Seg)

The time segments and also the nominal bit time are made up of integer units of time called time quanta or TQ. By definition, the nominal bit time has a minimum of 8 TQ and a maximum of 25 TQ. Also, by definition the minimum nominal bit time is 1 µsec, corresponding to a maximum 1 MHz bit rate.

**Figure 23-21: CAN Bit Timing**



### 23.9.2 Prescaler Setting

There is a programmable prescaler, with integral values ranging from 1 to 64, in addition to a fixed divide-by-2 for clock generation. The Time Quanta (TQ) is a fixed unit of time derived from the input clock period, TCAN. Time quanta is defined as:

**Equation 23-1: Time Quanta for Clock Generation**

$$TQ = 2 \bullet (BRP + 1) \bullet TCAN$$

Where BRP is the binary value of BRP <5:0>
TCAN is TCY or TCY/4 depending on CANCKS bit

**Example 23-1: Bit Rate Calculation Example**

If 4FCY = 32 MHz, BRP<5:0> = 0x01 and CANCKS = 0, then:

$$TQ = 2 \bullet (BRP + 1) \bullet \frac{TCY}{4} = 2 \text{ x } 2 \text{ x } (1/32\text{x}10^6) = 125\text{ns}$$

If Nominal Bit Time = 8 TQ then:
Nominal Bit Rate $= 1/(8 \times 125 \times 10^{-9})$ Mbps

**Example 23-2: Baud Rate Prescaler Calculation Example**

CAN Baud Rate = 125 kHz
FCY = 5 MHz, CANCKS = 1

1. Select number of TQ clocks per bit time (e.g., K=16).
2. Calculate TQ from baud rate:

$$TQ = \frac{1/(BaudRate)}{K} = \frac{1/125\times10^3}{16} = 500ns$$

3. Calculate BRP<5:0>:

$$TQ = 2 \bullet (BRP + 1) \bullet TCAN$$

$$BRP = (2TQ/TCY) - 1$$

$$= \frac{2(500\times10^{-9})}{1/(5\times10^6)} - 1$$

$$= 4$$

The frequencies of the oscillators in the different nodes must be coordinated in order to provide a system-wide specified time quantum. This means that all oscillators must have a TOSC that is a integral divisor of TQ.

### 23.9.3 Propagation Segment

This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus line and the internal delay time of the nodes. The delay is calculated as the round trip from transmitter to receiver as twice the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. The Propagation Segment can be programmed from 1 TQ to 8 TQ by setting the PRSEG<2:0> bits (CiCFG2<2:0>).

### 23.9.4 Phase Segments

The phase segments are used to optimally locate the sampling of the received bit within the transmitted bit time. The sampling point is between Phase1 Segment and Phase2 Segment. These segments are lengthened or shortened by re-synchronization. The end of the Phase1 Segment determines the sampling point within a bit period. The segment is programmable from 1 TQ to 8 TQ. Phase2 Segment provides delay to the next transmitted data transition. The segment is programmable from 1 TQ to 8 TQ or it may be defined to be equal to the greater of Phase1 Segment or the Information Processing Time (3 TQ's). The phase segment 1 is initialized by setting bits SEG1PH<2:0> (CiCFG2<5:3>), and phase segment 2 is initialized by setting SEG2PH<2:0> (CiCFG2<10:8>).

### 23.9.5 Sample Point

The sample point is the point of time at which the bus level is read and interpreted as the value of that respective bit. The location is at the end of phase segment 1. If the bit timing is slow and contains many TQ, it is possible to specify multiple sampling of the bus line at the sample point. The level determined by the CAN bus then corresponds to the result from the majority decision of three values. The majority samples are taken at the sample point and twice before with a distance of TQ/2. The CAN module allows to chose between sampling three times at the same point or once at the same point. This is done by setting or clearing the SAM bit (CiCFG2<6>).

### 23.9.6 Synchronization

To compensate for phase shifts between the oscillator frequencies of the different bus stations, each CAN controller must be able to synchronize to the relevant signal edge of the incoming signal. When an edge in the transmitted data is detected, the logic will compare the location of the edge to the expected time (Synchronous Segment). The circuit will then adjust the values of Phase1 Segment and Phase2 Segment. There are 2 mechanisms used to synchronize.

#### 23.9.6.1 Hard Synchronization

Hard Synchronization is only done whenever there is a "recessive" to "dominant" edge during bus Idle, indicating the start of a message. After hard synchronization, the bit time counters are restarted with Synchronous Segment. Hard synchronization forces the edge which has caused the hard synchronization to lie within the synchronization segment of the restarted bit time. Due to the rules of synchronization, if a hard synchronization is done, there will not be a re-synchronization within that bit time.

#### 23.9.6.2 Re-synchronization

As a result of re-synchronization phase segment 1 may be lengthened or phase segment 2 may be shortened. The amount of lengthening or shortening of the phase buffer segment, specified by the SJW<1:0> bits (CiCFG1<7:6>), has an upper bound given by the re-synchronization jump width bits. The value of the synchronization jump width will be added to phase segment 1 or subtracted from phase segment 2. The re-synchronization jump width is programmable between 1 TQ and 4 TQ.

Clocking information will only be derived from transitions of recessive to dominant bus states. The property that only a fixed maximum number of successive bits have the same value ensures resynchronizing a bus unit to the bit stream during a frame (e.g., bit-stuffing).

The phase error of an edge is given by the position of the edge relative to Synchronous Segment, measured in Time Quanta. The phase error is defined in magnitude of TQ as follows:

- e = 0 if the edge lies within the Synchronous Segment.
- e > 0 if the edge lies before the Sample Point.
- e < 0 if the edge lies after the Sample Point of the previous bit.

If the magnitude of the phase error is less than or equal to the programmed value of the re-synchronization jump width, the effect of a re-synchronization is the same as that of a hard synchronization.

If the magnitude of the phase error is larger than the re-synchronization jump width, and if the phase error is positive, then phase segment 1 is lengthened by an amount equal to the re-synchronization jump width.

If the magnitude of the phase error is larger than the re-synchronization jump width, and if the phase error is negative, then phase segment 2 is shortened by an amount equal to the re-synchronization jump width.

**Figure 23-22: Lengthening a Bit Period**

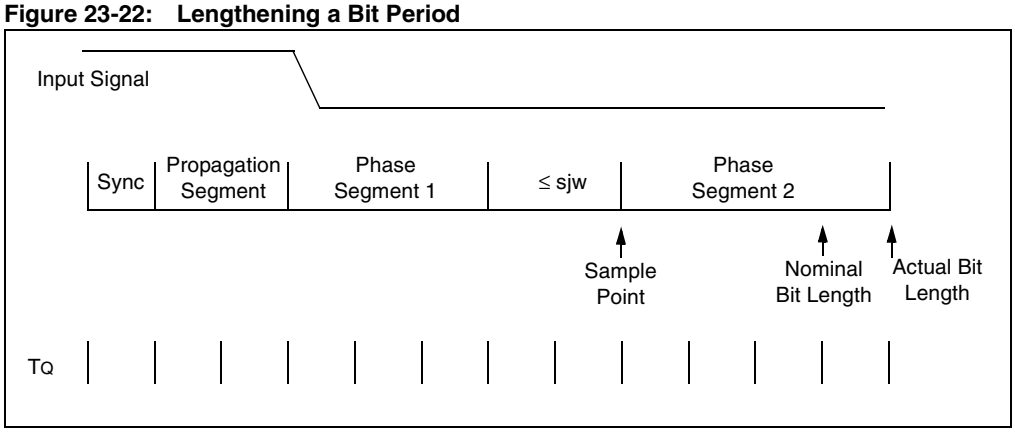
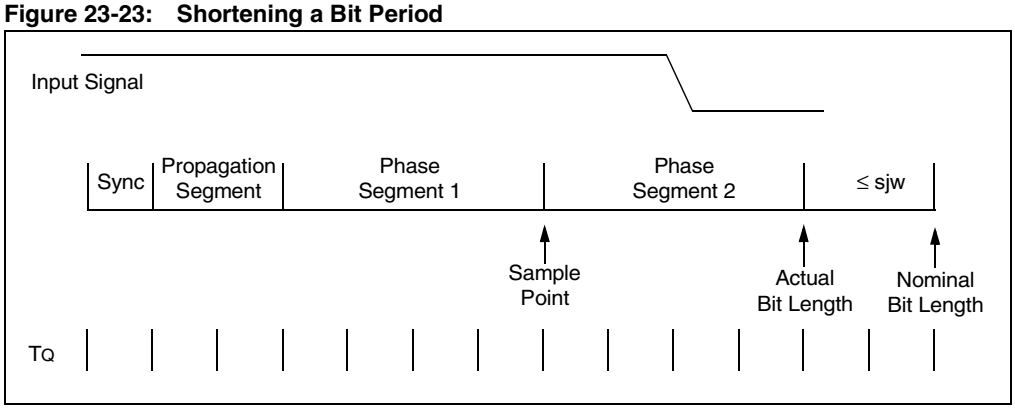



**Figure 23-23: Shortening a Bit Period**



### 23.9.7 Programming Time Segments

Some requirements for programming of the time segments are as follows:

- Propagation Segment + Phase1 Segment > = Phase2 Segment
- Phase2 Segment > Synchronous Jump Width

Typically, the sampling of the bit should take place at about 60-70% through the bit time, depending on the system parameters.

Example 23-2 has a 16 TQ bit time. If we choose Synchronous Segment = 1 TQ and Propagation Segment = 2 TQ, setting Phase Segment 1 = 7 TQ would place the sample point at 10 TQ (62% of the bit time) after the initial transition. This would leave 6 TQ for phase segment 2.

Since phase segment 2 is 6, according to the rules, the SJWS<1:0> bits could be set to the maximum of 4 TQ. However, normally a large synchronization jump width is only necessary when the clock generation of the different nodes is inaccurate or unstable, such as using ceramic resonators. So a synchronization jump width of 1 is typically enough.

## 23.10 Interrupts

The module has several sources of interrupts. Each of these interrupts can be individually enabled or disabled. A CiINTF register contains interrupt flags. A CiINTE register controls the enabling of the 8 main interrupts. A special set of read only bits in the CiCTRL register (ICODE<2:0>) can be used in combination with a jump table for efficient handling of interrupts.

All interrupts have one source, with the exception of the error interrupt. Any of the error interrupt sources can set the error interrupt flag. The source of the error interrupt can be determined by reading the CiINTF register.

The interrupts can be broken up into two categories: receive and transmit interrupts.

The receive related interrupts are:

- Receive interrupt
- Wake-up interrupt
- Receiver Overrun interrupt
- Receiver Warning interrupt
- Receiver Error Passive interrupt

The Transmit related interrupts are:

- Transmit interrupt
- Transmitter Warning interrupt
- Transmitter Error Passive interrupt
- Bus Off interrupt

### 23.10.1 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in CiINTF register. Interrupts are pending as long as one of the corresponding flags is set. The flags in the registers must be reset within the interrupt handler in order to handshake the interrupt. A flag can not be cleared if the respective condition still prevails, with the exception being interrupts that are caused by a certain value being reached in one of the error counter registers.

### 23.10.2 The ICODE Bits

The ICODE<2:0> bits (CiCTRL<3:1>) are a set of read only bits designed for efficient handling of interrupts via a jump table. The ICODE<2:0> bits can only display one interrupt at a time because the interrupt bits are multiplexed into this register. Therefore, the pending interrupt with the highest priority and enabled interrupt is reflected in the ICODE<2:0> bits. Once the highest priority interrupt flag has been cleared, the next highest priority interrupt code is reflected in the ICODE<2:0> bits. An interrupt code for a corresponding interrupt can only be displayed if both its interrupt flag and interrupt enable are set. Table 23-6 describes the operation of the ICODE<2:0> bits.

**Table 23-6: ICODE Bits Decode Table**

| ICODE<2:0> | Boolean Expression |
|---|---|
| 000 | $\overline{ERR}\bullet\overline{WAK}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet\overline{RX1}$ |
| 001 | ERR |
| 100 | $\overline{ERR}\bullet$TX0 |
| 011 | $\overline{ERR}\bullet\overline{TX0}\bullet$TX1 |
| 010 | $\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet$TX2 |
| 110 | $\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet$RX0 |
| 101 | $\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet$RX1 |
| 111 | $\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet\overline{RX1}\bullet$WAK |

Legend: ERR = ERRIF • ERRIE
TX0 = TX0IF • TX0IE
TX1 = TX1IF • TX1IE
TX2 = TX2IF • TX2IE
RX0 = RX0IF • RX0IE
RX1 = RX1IF • RX1IE
WAK = WAKIF • WAKIE

### 23.11 Time-stamping

The CAN module will generate a signal that can be sent to a timer capture input whenever a valid frame has been accepted. Because the CAN specification defines a frame to be valid if no errors occurred before the EOF field has been transmitted successfully, the timer signal will be generated right after the EOF. A pulse of one bit time is generated.

Time-stamping is enabled by the TSTAMP control bit (CiCTRL<15>). The capture input that is used for time-stamping depends on the device variant. Refer to the specific device data sheet for more information.

### 23.12 CAN Module I/O

The CAN bus module communicates on up to 2 I/O pins. There is 1 transmit pin and 1 receive pin. These pins are multiplexed with normal digital I/O functions of the device.

When the module is in the Configuration mode, Module Disable mode or Loopback mode, the I/O pins revert to the PORT I/O function.

When the module is active, the CiTX pin (i = 1 or 2) is always dedicated to the CAN output function. The TRIS bits associated with the transmit pins are overridden by the CAN bus modes. The module receives the CAN input on the CiRX input pin.

## 23.13 Operation in CPU Power Saving Modes

### 23.13.1 Operation in SLEEP Mode

SLEEP mode is entered by executing a `PWRSAV #0` instruction. This will stop the crystal oscillator and shut down all system clocks. The user should ensure that the module is not active when the CPU goes into SLEEP mode. The pins will revert into normal I/O function, dependent on the value in the TRIS register.

Because the CAN bus is not disruptable, the user must never execute a `PWRSAV #0` instruction while the module is in an Operating mode. The module must first be switched to Disable mode by setting REQOP<2:0> =`001` (CiCTRL<10:8>). When OPMODE<2:0> = `001` (CiCTRL<7:5>), indicating that Disable mode is achieved, then the SLEEP instruction may be used.

Figure 23-24 depicts how the CAN module will behave when the CPU enters SLEEP mode and how the module wakes up on bus activity. When the CPU exits SLEEP mode due to activity on the CAN bus, the WAKIF flag (CiINTF<6>) is set.

The module will monitor the CiRX line for activity while the device is in SLEEP mode.

If the device is in SLEEP mode and the WAKIE wake-up interrupt enable is set, the module will generate an interrupt, waking the CPU. Due to the delays in starting up the oscillator and CPU, the message activity that caused the wake-up will be lost.

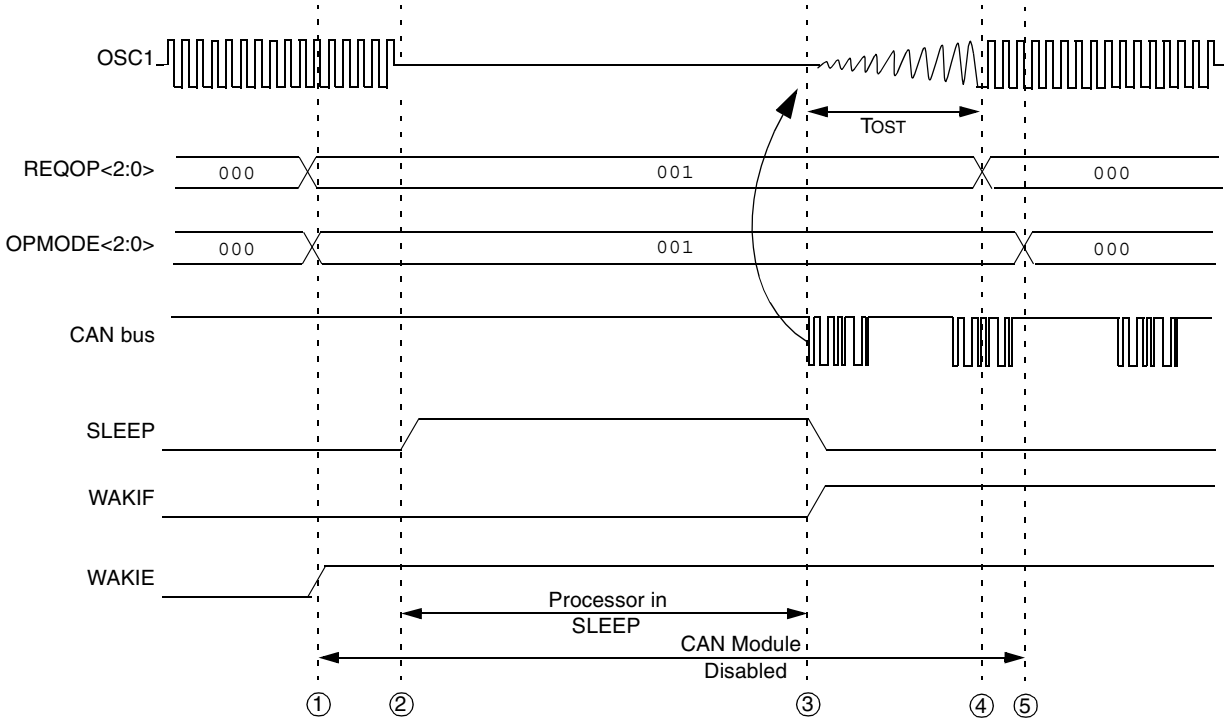
If the module is in CPU SLEEP mode and the WAKIE is not set, no interrupt will be generated and the CPU and the CAN module will continue to sleep.

If the CAN module is in Disable mode, the module will wake-up and, depending on the condition of the WAKIE bit, may generate an interrupt. It is expected that the module will correctly receive the message that caused the wake-up from SLEEP mode.

The module can be programmed to apply a low-pass filter function to the CiRX input line while the module or the CPU is in SLEEP mode. This feature can be used to protect the module from wake-up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic inference within noisy environments. The WAKFIL bit (CiCFG2<14>) enables or disables the filter.

**Figure 23-24: Processor SLEEP and CAN bus Wake-up Interrupt**




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

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

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

⑤ - Module detects 11 recessive bits. Module will begin to receive messages and transmit any pending messages.

### 23.13.2 CAN Module Operation during CPU IDLE Mode

On executing a CPU IDLE (`PWRSAV #1`) instruction, the operation of the CAN module is determined by the state of the CSIDL bit (CiCTRL<13>).

If CSIDL = `0`, the module will continue operation on assertion of IDLE mode. The CAN module can wake the device from IDLE mode if the CAN module interrupt is enabled.

If CSIDL = `1`, the module will discontinue operation in IDLE mode. The same rules and conditions for entry to and wake from SLEEP mode apply. Refer to **Section 23.13.1 "Operation in SLEEP Mode"** for further details.

### 23.14 CAN Protocol Overview

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a very high level of robustness. The CAN Protocol is fully defined by Robert Bosch GmbH, in the CAN Specification V2.0B from 1991.

Its domain of application ranges from high speed networks to low cost multiplex wiring. Automotive electronics (i.e., engine control units, sensors, anti-skid-systems, etc.) are connected using CAN with bit rates up to 1 Mbit/sec. The CAN Network allows a cost effective replacement of wiring harnesses in the automobile. The robustness of the bus in noisy environments and the ability to detect and recover from fault conditions makes the bus suitable for industrial control applications such as DeviceNet, SDS and other fieldbus protocols.

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus nodes. A CAN bus consists of two or more nodes. The number of nodes on the bus may be changed dynamically without disturbing the communication of other nodes. This allows easy connection and disconnection of bus nodes (e.g., for addition of system function, error recovery or bus monitoring).

The bus logic corresponds to a “wired-AND” mechanism, “recessive” bits (mostly, but not necessarily equivalent to the logic level ‘1’) are overwritten by “dominant” bits (mostly logic level ‘0’). As long as no bus node is sending a dominant bit, the bus line is in the recessive state, but a dominant bit from any bus node generates the dominant bus state. Therefore, for the CAN bus line, a medium must be chosen that is able to transmit the two possible bit states (dominant and recessive). One of the most common and cheapest ways is to use a twisted wire pair. The bus lines are then called “CANH” and “CANL”, and may be connected directly to the nodes, or via a connector. There's no standard defined by CAN regarding connector requirements. The twisted wire pair is terminated by terminating resistors at each end of the bus line. The maximum bus speed is 1 Mbit, which can be achieved with a bus length of up to 40 meters. For bus lengths longer than 40 meters, the bus speed must be reduced (a 1000 m bus can be realized with a 40 Kbit bus speed). For bus lengths above 1000 meters, special drivers should be used. At least 20 nodes may be connected without additional equipment. Due to the differential nature of transmission, CAN is not inherently susceptible to radiated electromagnetic energy because both bus lines are affected in the same way, which leaves the differential signal unaffected. The bus lines may also be shielded to reduce the radiated electromagnetic emission from the bus itself, especially at high baud rates.

The binary data is coded corresponding to the NRZ code (Non-Return-to-Zero; low level = dominant state; high level = recessive state). To ensure clock synchronization of all bus nodes, bit-stuffing is used. This means that during the transmission of a message, a maximum of five consecutive bits may have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (destuffing).

In the CAN protocol, it is not bus nodes that are addressed. The address information is contained in the messages that are transmitted. This is done via an identifier (part of each message) which identifies the message content (e.g., engine speed, oil temperature etc.). This identifier also indicates the priority of the message. The lower the binary value of the identifier, the higher the priority of the message.

For bus arbitration, Carrier Sense Multiple Access/Collision Detection (CSMA/CD) with Non-Destructive Arbitration (NDA) is used. If bus node A wants to transmit a message across the network, it first checks that the bus is in the IDLE state (“Carrier Sense”), (i.e., no node is currently transmitting). If this is the case (and no other node wishes to start a transmission at the same moment), node A becomes the bus master and sends its message. All other nodes switch to Receive mode during the first transmitted bit (Start-Of-Frame bit). After correct reception of the message (which is Acknowledged by each node), each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time (“Multiple Access”), collision of the messages is avoided by bitwise arbitration (“Collision Detection/Non-Destructive Arbitration” together with the “Wired-AND” mechanism, “dominant” bits override “recessive” bits). Each node sends the bits of its message identifier (MSb first) and monitors the bus level. A node that sends a recessive identifier bit but reads back a dominant one loses bus arbitration and switches to Receive mode. This condition occurs when the message identifier of a competing node has a lower binary value (dominant state = logic 0) and therefore, the competing node is sending a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. All other nodes automatically try to repeat their transmission once the bus returns to the IDLE state. It is not permitted for different nodes to send messages with the same identifier, as arbitration could fail, leading to collisions and errors later in the message.

The original CAN specifications (Versions 1.0, 1.2 and 2.0A) defined the message identifier as having a length of 11 bits giving a possible 2048 message identifiers. The specification has since been updated (to version 2.0B) to remove this limitation. CAN specification Version 2.0B allows message identifier lengths of 11 and/or 29 bits to be used (an identifier length of 29 bits allows over 536 million message identifiers). Version 2.0B CAN is also referred to as “Extended CAN”; and Versions 1.0, 1.2 and 2.0A) are referred to as “Standard CAN”.

### 23.14.1 Standard CAN vs. Extended CAN

Those data frames and remote frames, which only contain the 11-bit identifier, are called standard frames according to CAN specification V2.0A. With these frames, 2048 different messages can be identified (identifiers 0-2047). However, the 16 messages with the lowest priority (2032-2047) are reserved. Extended frames according to CAN specification V2.0B have a 29-bit identifier. As already mentioned, this 29-bit identifier is made up of the 11-bit identifier (“Standard ID”) and the 18-bit Extended identifier (“Extended ID”).

CAN modules specified by CAN V2.0A are only able to transmit and receive standard frames according to the Standard CAN protocol. Messages using the 29-bit identifier cause errors. If a device is specified by CAN V2.0B, there is one more distinction. Modules named “Part B Passive” can only transmit and receive standard frames but tolerate extended frames without generating error frames. “Part B Active” devices are able to transmit and receive both standard and extended frames.

### 23.14.2 ISO Model

The ISO/OSI Reference Model is used to define the layers of protocol of a communication system as shown in Figure 23-25. At the highest end, the applications need to communicate between each other. At the lowest end, some physical medium is used to provide electrical signaling.

The higher levels of the protocol are run by software. Within the CAN bus specification, there is no definition of the type of message, or the contents, or meaning of the messages transferred. These definitions are made in systems such as Volcano, the Volvo automotive CAN specification J1939, the U.S. heavy truck multiplex wiring specification; and Allen-Bradley DeviceNet and Honeywell SDS, industrial protocols.

The CAN bus module definition encompasses two levels of the overall protocol:

- The Data Link Layer
  - The Logical Link Control (LLC) sub layer
  - The Medium Access Control (MAC) sub layer
- The Physical Layer
  - The Physical Signaling (PLS) sub layer

The LLC sub layer is concerned with Message Filtering, Overload Notification and Error Recovery Management. The scope of the LLC sub layer is:

- To provide services for data transfer and for remote data request.
- To decide which messages received by the LLC sub layer are actually to be accepted.
- To provide means for error recovery management and overload notifications.

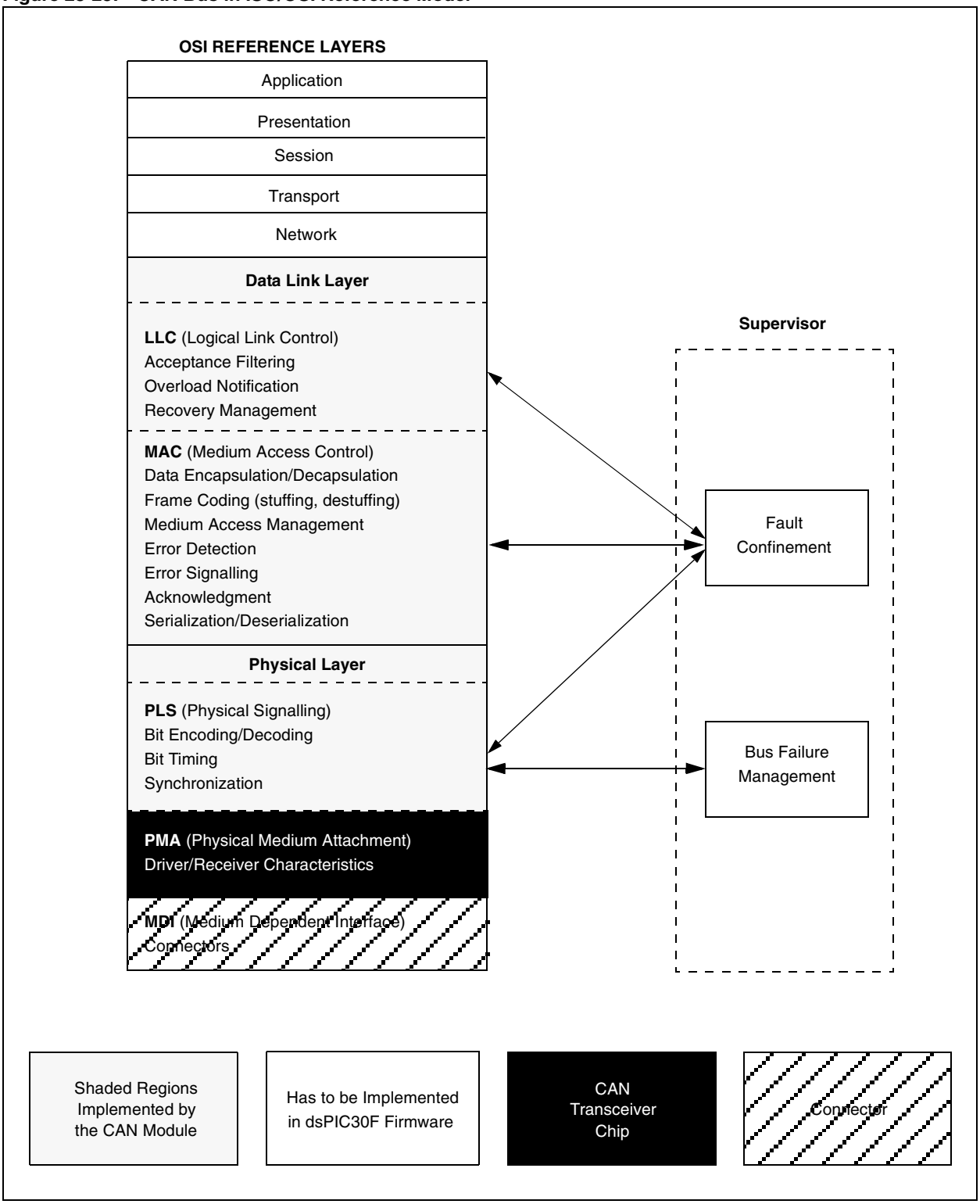
The MAC sub layer represents the kernel of the CAN protocol. The MAC sub layer defines the transfer protocol, (i.e., controlling the Framing, Performing Arbitration, Error Checking, Error Signalling and Fault Confinement). It presents messages received from the LLC sub layer and accepts messages to be transmitted to the LLC sub layer. Within the MAC sub layer is where it's decided whether the bus is free for starting a new transmission, or whether a reception is just starting. The MAC sub layer is supervised by a management entity called Fault Confinement which is a self-checking mechanism for distinguishing short disturbances from permanent failures. Also, some general features of the bit timing are regarded as part of the MAC sub layer.

The physical layer defines the actual transfer of the bits between the different nodes with respect to all electrical properties. The PLS sub layer defines how signals are actually transmitted and therefore deals with the description of Bit Timing, Bit Encoding and Synchronization.

The lower levels of the protocol are implemented in driver/receiver chips and the actual interface such as twisted pair wiring or optical fiber, etc. Within one network, the physical layer has to be the same for all nodes. The driver/receiver characteristics of the physical layer are not defined in the CAN specification so as to allow transmission medium and signal level implementations to be optimized for their application. The most common example of the physical transmission medium is defined in Road Vehicles ISO11898, a multiplex wiring specification.

**Figure 23-25: CAN Bus in ISO/OSI Reference Model**



## 23.15 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC30F Product Family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the CAN module are:

| Title | Application Note # |
|---|---|
| An Introduction to the CAN Protocol | AN713 |

**Note:** Please visit the Microchip web site (www.microchip.com) for additional Application Notes and code examples for the dsPIC30F Family of devices.

## 23.16 Revision History

### Revision A

This is the initial released revision of this document.

### Revision B

This revision incorporates additional technical content for the dsPIC30F CAN module.

**NOTES:**