AN2283/D
Rev. 0, 8/2002

a Scalable Controller Area Network (MSCAN) Interrupts

By: Shaila G. Konanahalli
A.J. Pohlmeyer
Field Application Engineers
Detroit Sales Office

Introduction

This application note is intended to help the user understand the Freescale **S**calable **C**ontroller **A**rea **N**etwork (MSCAN) interrupts on HC08 and HC12 devices; however, does not apply to the MSCAN on the HCS12. The MSCAN interrupt behavior is identical on both the HC08 and HC12 implementations. This application note describes the interrupts MSCAN interrupts in detail and explains why they are used, what they are used for, and when to use them. After reading this application note one should be able to clearly understand the various MSCAN interrupts and how they should be implemented in their application.

Basic Overview

The MSCAN supports four interrupt vectors on which are mapped 11 different interrupt sources, of which any of the 11 can be individually masked. By implementing the four separate vectors, the interrupt source can be easily identified reducing interrupt service time. These interrupt vectors include from highest priority to lowest priority:

- Wakeup
- Error
- Receive
- Transmit

The error interrupt vector consists of six different error interrupt sources which are shown in **Table 1**.

Table 1. Error Interrupt Sources

Interrupt Vector	Sources
Wakeup	Wakeup interrupt
Error	Overrun interrupt Bus-off interrupt Transmitter error passive interrupt Receiver error passive interrupt Transmitter warning interrupt Receiver warning interrupt
Receive	Receive buffer full interrupt
Transmit	Transmit buffer 0 empty interrupt Transmit buffer 1 empty interrupt Transmit buffer 2 empty interrupt

Wakeup Interrupt

While the MSCAN is in Sleep Mode, it will wakeup when bus activity is detected. The central processor unit (CPU) requests the MSCAN to go into Sleep Mode after the bus traffic has been halted, typically signaled by a system-level sleep command message. Once the network is asleep, a wakeup interrupt will be needed when message traffic resumes to bring the node out of Sleep Mode.

For further information, refer to *MSCAN Low-Power Applications* (Freescale document order number AN2255/D).

Wakeup Interrupt Initialization

Sleep Mode is used to reduce the power consumption in the system. The CPU requests the MSCAN to enter Sleep Mode by setting the Sleep Request (SLPRQ) bit in the MSCAN Module Control Register 0 (CMCR0) register (see **Figure 1**).

The MSCAN enters Sleep Mode depending on its current activity:

- If transmitting, it will continue transmitting until there are no more messages to be transmitted and then enter the Sleep Mode.
- If receiving, it will wait for the end of the message and then enter the Sleep Mode.
- If neither transmitting nor receiving, it will enter the Sleep Mode immediately.

NOTE: *When the MSCAN is running, the control bit SLPRQ = 0 and the status bit Sleep Mode Acknowledge (SLPAK) = 0 (these bits are located in the CMCR0 register).*

Steps to enter Sleep Mode:

1. The first step for entering Sleep Mode is to abort all pending transmissions. See **Abort Process Overview**.
2. The CPU requests the Sleep Mode by setting the control bit SLPRQ = 1 in the CMCR0 register.
3. The MSCAN acknowledges entering Sleep Mode by setting the status bit SLPAK = 1.
4. Next, enable the wakeup interrupt by setting the Wakeup Interrupt Enable (WUPIE) bit = 1 in the MSCAN Receiver Interrupt Enable Register (CRIER).

NOTE: *The CPU must wait for SLPAK = 1 before it can execute its STOP instruction. This may take time depending on steps 1 or 2.*

Steps to exit Sleep Mode:

1. When bus activity is detected:
   a. The MSCAN clears the control bit SLPRQ and the status bit SLPAK in the CMCR0 register.
   b. The CPU wakeup only occurs if WUPIE is set and the I bit of the CCR register is clear. The Wakeup Interrupt Flag (WUPIF) bit will be set in the MSCAN Receiver Flag Register (CRFLG), which in turn triggers the CPU to wakeup from its Stop Mode.
   c. The clock sources will not be stable immediately out of Stop Mode. Therefore, the first message will probably be missed.

or

2. When an external interrupt of the host CPU (for example, IRQ, keyboard, etc.) occurs, the CPU needs to clear the control bit SLPRQ (Sleep Request) bit in the CMCR0 register.

The CMCR0 and CRFLG registers are shown in **Figure 1** and **Figure 2**.

Bit 7	6	5	4	3	2	1	Bit 0
0	0	CSWAI	SYNCH	TLNKEN	SLPAK	SLPRQ	SFTRES

Figure 1. MSCAN Module Control Register 0 (CMCR0)

Bit 7	6	5	4	3	2	1	Bit 0
WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF

Figure 2. MSCAN Receiver Flag Register (CRFLG)

Transmit Structure Overview

This subsection discusses the transmit structure on the MSCAN08 and MSCAN12. The MSCAN has three transmit buffers so that multiple messages can be set up for transmission onto the CAN bus. When more than one transmit message buffer is filled with a message, the local priority values in the corresponding Transmit Buffer Priority Register (TBPR) are compared to determine which message is to be transmitted onto the bus first.

Each transmit buffer consists of a 14-byte data structure:

- Four Identifier Registers
- Eight Data Segment Registers
- One Data Length Register
- One Transmit Buffer Priority Register

See **Table 2**.

Table 2. Data Structure

Register Name
Identifier Register 0
Identifier Register 1
Identifier Register 2
Identifier Register 3
Data Segment Register 0
Data Segment Register 1
Data Segment Register 2
Data Segment Register 3
Data Segment Register 4
Data Segment Register 5
Data Segment Register 6
Data Segment Register 7
Data Length Register
Transmit Buffer Priority Register

Identifier Registers

The Identifier Registers consists of either 11 bits (ID10–ID0) for the standard CAN 2.0 A/B format or 29 bits (ID28–ID0) for the extended CAN2.0 B format. See **Figure 3** and **Figure 4**

	Bit 7	6	5	4	3	2	1	Bit 0
Identifier Register 0 (IDR0)	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
Identifier Register 1 (IDR1)	ID2	ID1	ID0	*RTR*	*IDE*			
Identifier Register 2 (IDR2)								
Identifier Register 3 (IDR3)								

Figure 3. Standard Identifier Registers

	Bit 7	6	5	4	3	2	1	Bit 0
Identifier Register 0 (IDR0)	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
Identifier Register 1 (IDR1)	ID20	ID19	ID18	*SRR*	*IDE*	ID17	ID16	ID15
Identifier Register 2 (IDR2)	ID24	ID13	ID12	ID11	ID10	ID9	ID8	ID7
Identifier Register 3 (IDR3)	ID6	ID5	ID4	ID3	ID2	ID1	ID0	*RTR*

Figure 4. Extended Identifier Registers

SRR — Substitute Remote Request

This is a fixed recessive bit [recessive bit = 1 and dominant bit = 0] used only in the extended format mode. It must be set to 1 by the user for transmission buffers and will be stored as received on the CAN bus for receive buffers.

IDE — ID Extended

This flag indicates whether the extended or standard identifier format is applied in this buffer.

1 = Extended format (29 bit)
0 = Standard format (11 bit)

NOTE: *In the case of IDE = 0, ID17–ID0 and the RTR bits are not applicable.*

RTR — Remote Transmission Request

This flag defines the type of message frame that will be transmitted onto the CAN bus and whether or not that frame contains data. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent.

1 = Remote frame
0 = Data frame

Data Length Register

The Data Length Register contains information on the number of data bytes in the CAN message. The data byte count ranges from 0–8 for the data frame (see **Table 3**).

Table 3. Data Length Code and Byte Count

Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

Data Segment Registers

The eight Data Segment Registers contain the data bytes to be transmitted. The number of bytes being transmitted is determined by the data length code in the data length register (see **Table 3**).

Transmit Buffer Priority Register

Bit 7	6	5	4	3	2	1	Bit 0
PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0

Figure 5. Transmit Buffer Priority Register (TBPR)

PRIO7–PRIO0 — Local Priority

This field defines the local priority of the associated message buffer. The local priority is used for the internal prioritization of the transmit process of the MSCAN and is defined to be highest for the smallest binary value.

- All transmission buffers with cleared Transmitter Buffer Empty (TXE) flags in the CAN Transmitter Flag Register (CTFLG) participate in the prioritization immediately before the Start of Frame (SOF) is sent.
- The transmission buffer with the lowest local priority field wins prioritization.
- If more than one buffer has the same lowest priority value, the message buffer with the lower index wins. Proper assignment of priority levels should be done at design (referring to the message ID strategy) to prevent this from possibly changing the intended priority of messages transmitted.

In summary, a message ready for transmission consists of an Identifier (ID), Data Length Code (DLC), Data, and the Transmit Buffer Priority Register (TBPR). A Transmit Header Table (see **Table 4**) can be implemented consisting of the ID, the DLC, and a pointer to the data (*Data_Pointer) of each particular message to be transmitted from this node (ordered from highest priority to lowest in the table). The index value into the table would then be used as a local priority for each message to be transmitted. This technique will ensure that each message will have a unique “local priority”. Messages transmitted from the same node should have different “local priorities” (TBPR) assigned to them so messages will not lose their precedence over one another when it comes time to transmit.

Table 4. Transmit Header Table

		ID	DLC	*Data_Pointer	
Index into table becomes priority value →	0				Highest Priority Message
	N				Lowest Priority Message

A structure can be created to refer to a particular row. The following example structure shows a standard identifier being used.

```
/******* Transmit Header Table ***********/

typedef struct
{
volatile unsigned int ID:  16;
volatile unsigned char DLC: 4;
volatile unsigned int *Data_Pointer;

} Transmit_Header;

Transmit_Header  Tx_Table [N];  // This line refers to the
                                // number of rows (the number
                                // of messages N+1) in
                                // the table.
```

Transmission Process Overview

Building on a good understanding of the transmit structure of the MSCAN, this subsection will address the transmitting process of the MSCAN. There are two registers involved in the transmit process:

- MSCAN Transmitter Control Register (CTCR)
- MSCAN Transmitter Flag Register (CTFLG)

The CTCR register consists of three bits: TXEIE2, TXEIE1, and TXEIE0 which are the Transmitter Empty Interrupt Enable bits corresponding to the three transmit buffers. If the user wishes to generate an interrupt to signal when a transmit buffer has been emptied after a successful transmission, the bits TXEIE2–TXEIE0 need to be enabled. This topic will be covered more in depth in **Transmission Process**.

The CTFLG register consists of three bits: TXE2, TXE1, and TXE0 that are the Transmitter Buffer Empty bits for the corresponding three transmit buffers. These bits will be set to 1 by the MSCAN when the associated message buffer is emptied. When this occurs, the user can load a new message into the empty transmit buffer. When a new message is ready for transmission, the associated TXEx bit in the CTFLG register is cleared by the user changing the buffer state to “transmit pending”. If more than one transmit buffer is ready for transmission, the “local priority” of the buffers will determine which of them will be transmitted first.

The MSCAN will now decide when the actual transmission onto the CAN bus occurs and will perform the necessary arbitration and error checking.

For example, let's transmit these three messages at the same time:

Message: ID = 0x0230, TBPR = 0x02;
Message: ID = 0x0016, TBPR = 0x00;
Message: ID = 0x072F, TBPR = 0x01;

When more than one buffer is loaded with a message, internal prioritization of the MSCAN occurs. The TBPR ("local priority") determines which buffer will transmit first. The lower the TBPR the higher the priority, in the same manner as arbitration of IDs on the bus.

In the example (**Figure 6**), all three buffers will contain the above three message IDs. Looking at their respective TBPRs, the buffer with TBPR = 0x00 will transmit first, then the buffer with TBPR = 0x01, and lastly the buffer with TBPR = 0x02.

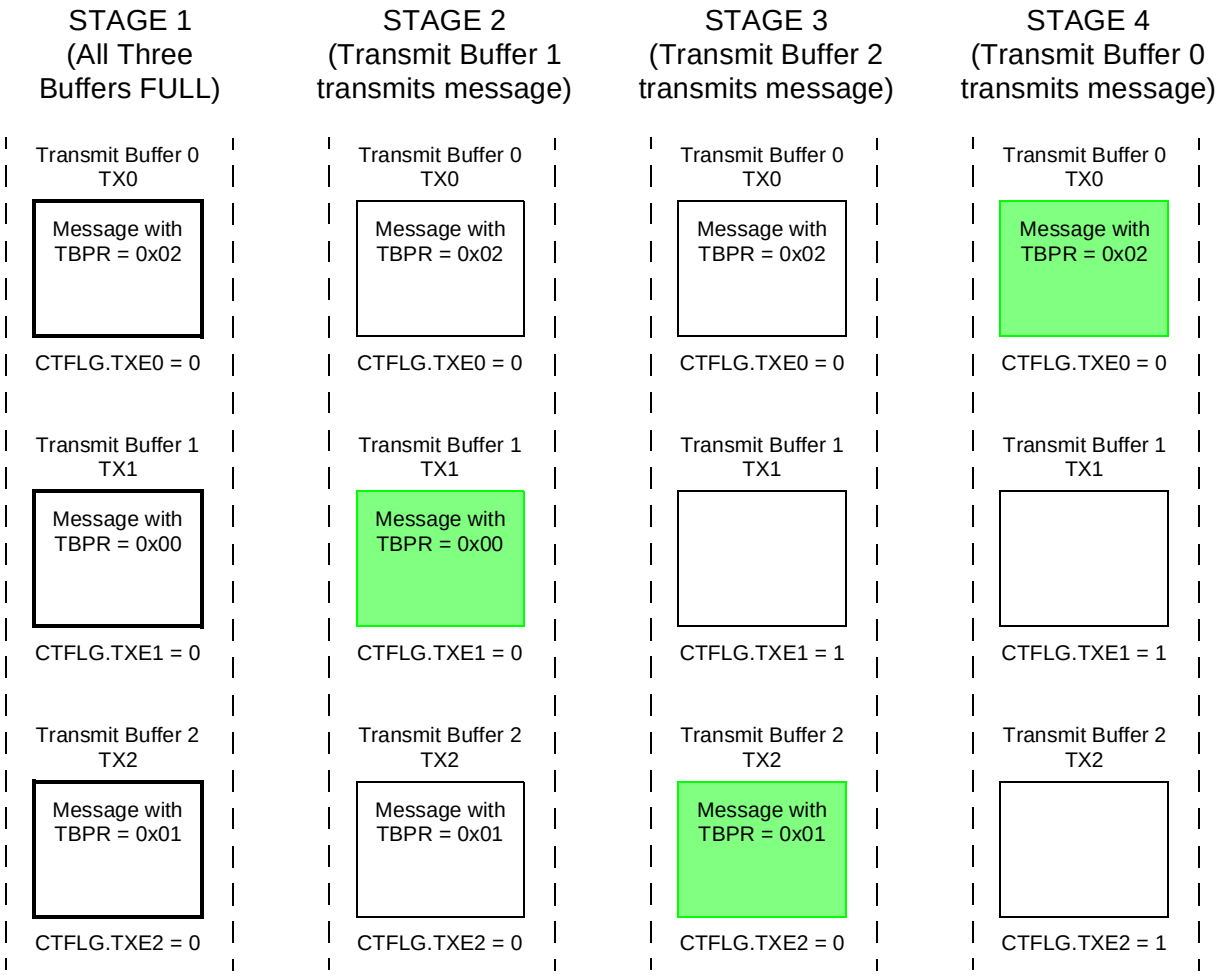


Figure 6. Buffer Transmission Example

Transmission Process

The MSCAN Transmitter Flag Register (CTFLG) and the MSCAN Transmitter Control Register (CTCR) are relevant to the transmit process on the MSCAN. See **Figure 7** and **Figure 8**.

Bit 7	6	5	4	3	2	1	Bit 0
0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0

Figure 7. MSCAN Transmitter Flag Register (CTFLG)

NOTE: *This CTFLG register is a special one. ABTAK2–ABTAK0 are status bits; however, TXE2–TXE0 are status as well as control bits.*

ABTAK2–ABTAK0 — Abort Acknowledge

See **Abort Process Overview**.

TXE2–TXE0 — Transmit Buffer Empty

This flag indicates that the associated transmit message buffer is empty; therefore, not scheduled for transmission. It is cleared by the user writing a 1 (using the exclusive-OR operation: $1 \oplus 1 = 0$) when a message has been loaded in the transmit buffer and is due for transmission. The MSCAN will set this flag after the message has been sent successfully, or when a message set up for transmission in a buffer is successfully aborted due to an abort request (see **Abort Process Overview**).

1 (set) = The associated message buffer is empty (not scheduled).

0 (clear) = The associated message buffer is filled and is scheduled for transmission.

Bit 7	6	5	4	3	2	1	Bit 0
0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0

Figure 8. MSCAN Transmitter Control Register (CTCR)

ABTRQ2–ABTRQ0 — Abort Request

See **Abort Process Overview**.

TXEIE2–TXEIE0 — Transmitter Empty Interrupt Enable

The user sets these bits while the buffer is filled. When the associated transmit buffers are empty (TXEx = 1) after a successful transmission, the interrupt occurs indicating a message can be loaded into that empty buffer.

1 (set) = When the corresponding transmit buffer is emptied it will result in a transmitter empty interrupt.

0 (clear) = No interrupt will be generated when the corresponding transmit buffer is empty.

Steps involved in the transmit process using the transmit interrupt are:

1. Find an empty transmit buffer as indicated by at least one of the transmit buffer empty flags (TXE2, TXE1, and TXE0) set to 1 in CTFLG.
2. Load the message contents (ID, DLC, Data, and Local Priority) into that empty transmit buffer.
3. Clear the corresponding transmit buffer's TXE flag to 0 indicating that the buffer is full. Keep in mind that these bits operate under the "exclusive-OR" operation. When TXE = 1 (transmit buffer empty), write a 1 to get TXE = 0 (transmit buffer full).

WARNING: *Do not use the BSET instruction, or you will clear ALL the TXE bits. This is due to the read, modify, and write operation of the BSET instruction. Instead LDA and STA instructions or MOVE should be used. If programming in C, ensure that the compiler is not performing the BSET instruction to perform these operations.*

4. Enable the transmit interrupt for that buffer. To do this, set the corresponding TXEIE bit to 1 in CTCR. The transmit interrupt will occur when the state changes from TXE = 0 (buffer full) to TXE = 1 (buffer empty). This occurs when the transmission of the message in the buffer is completed with no errors and an acknowledgement is received.
5. In the transmit interrupt routine, the user disables the transmit interrupt by writing a 0 to the corresponding TXEIE that caused the interrupt in the CTCR.

The following sample code illustrates the above steps.

```
/*******************************************************/
                TRANSMITTER INTERRUPT
/*******************************************************/
@interrupt void trasmit_interrupt(void){


    volatile unsigned char k;
    volatile unsigned char length;

    //Check to see if Transmit Buffer 0 is Empty
    if (ctflg.TXE0 == 1){
      ctcr.TXEIE0 = 0;  // Disable the transmit interrupt that caused the interrupt
       tx_buffer0.ID0 = Tx_Table[index].ID;  // Load the the message's ID from the Transmit Table into the Transmit Buffer
       tx_buffer0.DLC = Tx_Table[index].DLC; //Load the message's DLC from the Transmit Table into the Transmit Buffer
       length = tx_buffer0.DLC;  //Store the Data Length of the message into a variable to be used in the next step

      // Go to the address pointed to by the message's *Data_Pointer from the Transmit Table and load in the data
      // The number of data bytes in the message is loaded which is determined by length
       for(k=0; k<length; ++k){tx_buffer0.data[k] = *Tx_Table[index].Data + k;}

      //Load the priority into the transmit buffer.  The priority is the index value to the message in the Transmit Table.
      tx_buffer0.Priority = index;

      CTFLG = 0x01;  //Clear the TXE flag in the CTFLG register to indicate this buffer is full

      //The following is another method to clear the TXE flag to indicate this buffer is full in assembly language
      //Notice that the LDA and STA instructions are used. DO NOT USE THE BSET Instructions.
```

```
        /*
        #asm
           LDAA #0x01;
           STAA $0x106;
        #endasm
        */

  ctcrr.TXEIE0 = 1; //Enable the transmit interrupt for this buffer

                  } // End of if (ctflg.TXE0 == 1)

//Check to see if Transmit Buffer 1 is Empty
else if(ctflg.TXE1 == 1){
  ctcr.TXEIE1 = 0; //Disable the transmit interrupt that caused the interrupt
  tx_buffer1.ID0 = Tx_Table[index].ID; //Load the message's ID from the Transmit Table into the Transmit Buffer
   tx_buffer1.DLC = Tx_Table[index].DLC;  // Load the message's DLC from the Transit Table into the Transmit Buffer

  length = tx_buffer1.DLC; //Store the Data Length of the message into a variable to used in the next step

  // Go to the address pointed to by the message's *Data_Pointer from the Transmit Table and load in the data
  // The number of data bytes in the message is loaded which is determined by length
  for(k=0; k<length; ++k){tx_buffer0.data[k] = *Tx_Table[index].Data + k;}

  //Load the priority into the transmit buffer.  The priority is the index value to the message in the Transmit Table.
  tx_buffer1.Priority = index;

  CTFLG = 0x02;  //Clear the TXE flag in the CTFLG register to indicate this buffer is full

  //The following is another method to clear the TXE flag to indicate this buffer is full in assembly language
  //Notice that the LDA and STA instructions are used.  DO NOT USE THE BSET Instructions.
        /*
        #asm
           LDAA #0x02;
           STAA $0x106;
        #endasm
        */

  ctcrr.TXEIE1 = 1; //Enable the transmit interrupt for this buffer

                      } // End of if (ctflg.TXE1 ==  1)



//Check to see if Transmit Buffer 2 is Empty
else if(ctflg.TXE2 == 1){
   ctcr.TXEIE2 = 0;  //Disable the transmit interrupt that caused the interrupt
   tx_buffer2.ID0 = Tx_Table[index].ID; //Load the message's ID from the Transmit Table into the Transmit Buffer
   tx_buffer2.DLC = Tx_Table[index].DLC; // Load the message's DLC from the Transit Table into the Transmit Buffer

  length = tx_buffer2.DLC; //Store the Data Length of the message into a variable to used in the next step

  // Go to the address pointed to by the message's *Data_Pointer from the Transmit Table and load in the data
  // The number of data bytes in the message is loaded which is determined by length
  for(k=0; k<length; ++k){tx_buffer0.data[k] = *Tx_Table[index].Data + k;}
  //Load the priority into the transmit buffer.  The priority is the index value to the message in the Transmit Table.
  tx_buffer2.Priority = index;

  CTFLG = 0x04;   //Clear the TXE flag in the CTFLG register to indicate this buffer is full

  //The following is another method to clear the TXE flag to indicate this buffer is full in assembly language
  //Notice that the LDA and STA instructions are used.  DO NOT USE THE BSET Instructions.
        /*
        #asm
           LDAA #0x04;
           STAA $0x106;
           #endasm
        */
```

```
  ctcr.TXEIE2 = 1; //Enable the transmit interrupt for this buffer


                  // End of if (ctflg.TXE2 ==  1)

  //Check to see if Transmit Buffer 2 is Empty
else if(ctflg. TXE2 = = 1){
   ctcr.TXEIE2 = 0; //Disable the transmit interrupt that caused the interrupt
   tx_buffer2.IDO = Tx_Table[index].ID;  //Load the message's ID from the Transmit Table into the Transmit Buffer
   tx_buffer2.DLC = Tx_Table[index].DLC,  //Load the message's DLC from the Transmit Table into the Transmit Buffer

   length = tx_buffer2.DLC;  //Store the Data Length of the message into a variable to use in the next step

  //Go to the address pointed to by the message's *Data_Pointer from the Transmit TAble and load in the data
  //The number of data bytes in the message is loaded which is determined by length
  for(k=0; k<length; ++k) {tx_buffer0.data[k] = *Tx_Table[index].Data + k;}

  //Load the priority into the transmit buffer. The priority is the index value to the message in the Transmit Table
  tx_buffer2.Priority = index;

  CTFLG = 0x04;  //Clear the TXE flag in the CTFLG register to indicate this buffer is full

  //The following is another method to clear the TXE flag to indicate this buffer is full in assembly language
  //Notice that the LDA and STA instructions are used. DO NOT USE THE BSET Instructions.
       /*
       #asm
          LDAA #0x04;
          STAA $0x106;
       #endasm
       */

  ctcr.TXEIE2 = 1;  //Enable the transmit interrupt for this buffer


          } //End of if(ctflg.TXE2 == 1)


} // End of Transmit Interrupt
```

As seen in the code, using the transmit header table allows an efficient method to load the identifier/DLC data from the table to the transmit buffers one for one. The *Data_Pointer in the table references message signals that are stored in data structures that map to the MSCAN transmit buffers Data Field. This type of data storage improves driver efficiency and execution time.

NOTE: *The transmit interrupt routine is short, just a couple steps. This is very important to achieve code/time efficiency.*

Abort Process Overview

The MSCAN has the capability to abort a message that is in the transmission pending state. There are two registers that are used to abort messages in transmission:

- MSCAN Transmitter Flag Register (CTFLG)
- MSCAN Transmitter Control Register (CTCR)

In the CTCR register there are the ABTRQ2, ABTRQ1, and ABTRQ0 bits which are the Abort Request bits corresponding to the three transmit buffers respectively. The user will need to set the appropriate ABTRQx bit to abort the message in the corresponding transmit buffer.

NOTE: *A message currently transmitting on the bus CANNOT be aborted unless an error or loss of arbitration occurs during transmission.*

Bit 7	6	5	4	3	2	1	Bit 0
0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0

Figure 9. MSCAN Transmitter Flag Register (CTFLG))

ABTAK2–ABTAK0 — Abort Acknowledge

This flag acknowledges that a message in the corresponding buffer has been aborted from transmission due to a pending abort request from the user. After a particular message buffer has been flagged empty (TXE2, TXE1, TXE0 = 1), this flag can be used by the application software to identify whether or not the message has been aborted successfully.

1 (set) = The message has been aborted
0 (clear) = The message has not been aborted, message was sent successfully

TXE2–TXE0 — Transmit Buffer Empty

See **Transmission Process Overview**.

Bit 7	6	5	4	3	2	1	Bit 0
0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0

Figure 10. MSCAN Transmitter Control Register (CTCR)

ABTRQ2–ABTRQ0 — Abort Request

The user sets this bit to request that an already scheduled message buffer (TXE = 0 = buffer is full) be aborted. MSCAN will grant the request when the message is not already under transmission. When a message is aborted, the associated TXE and the Abort Acknowledge Flag (ABTAK) will be set and a TXE interrupt will occur if enabled.

1 (set) = Abort request is pending
0 (clear) = No abort request

TXEIE2–TXEIE0 — Transmitter Empty Interrupt Enable

See **Transmission Process Overview**.

Abort Request

An abort request of a message scheduled for transmission may be necessary:

1. When a message in a buffer is taking too long to transmit it eventually times out (from the user's application). This is because either it has a lower priority relative to the other messages in the other two message buffers or message traffic has higher priority resulting in loss of arbitration with the bus. The user's application code may elect to monitor the time a message is queued in a transmit buffer and remove the message if the data is determined to be stale.
2. When a state change occurs in a module that requires an abort of a pending message, with an updated message to be sent in its place.
3. When all three message buffers are filled with messages to be scheduled for transmission and a higher priority message (relative to the other messages) needs to be transmitted.

Figure 11 shows the steps taken by the MSCAN when an abort of a message is requested (ABTRQx = 1)

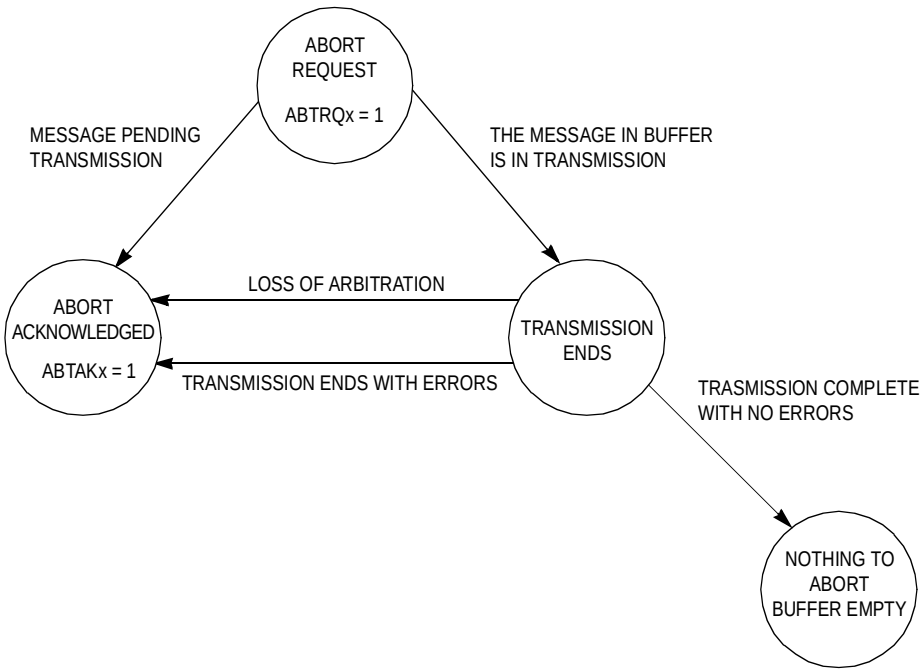


Figure 11. Abort Message Request Flowchart

As seen in **Figure 11**, when there is an abort request for a message there are four separate scenarios:

1. When the message in the buffer is not undergoing transmission on the bus the abort is granted immediately, ABTAKx = 1.
2. When the message in the buffer is undergoing transmission on the bus the MSCAN waits until the transmission completes. Transmission can complete due to:
   a. Loss of arbitration — the message is aborted and ABTAK = 1 (Abort Acknowledged)
   b. Transmission ends with errors — the message is aborted and ABTAK = 1 (Abort Acknowledged)
   c. Transmission is complete with no errors — in this case, there is nothing to abort because the MSCAN tried to abort a now empty buffer, so ABTAKx = 0 (Abort Not Acknowledged)

The transmit interrupt routine may need to be modified to support the abort capability in the MSCAN. For example, when a transmit empty buffer is detected the service routine needs to signal the network/transport software layers if the transmission completed successfully or the message was aborted. One method is to issue a completion code to a transmit status routine

Receive Structure Overview

To understand the receive process, the first topic to cover is the receive structure on the MSCAN. The MSCAN receive structure consists of a two-stage input FIFO (First In First Out) buffer structure. The MSCAN validates the received message and if no errors are detected, the message then goes through a hardware filtering process of the identifier field to determine whether to save the message in the receive FIFO or discard it.

The MSCAN's two receive buffers are alternately mapped into a single location in the memory map. The received messages are stored in a two-stage input FIFO buffer structure:

- The Background Receive Buffer (RxBG) is exclusively accessible by the MSCAN
- The Foreground Receive Buffer (RxFG) is accessible by the CPU

Both the RxFG and RxBG buffers have a 13-byte data structure similar to the transmit buffer. See **Figure 12**.

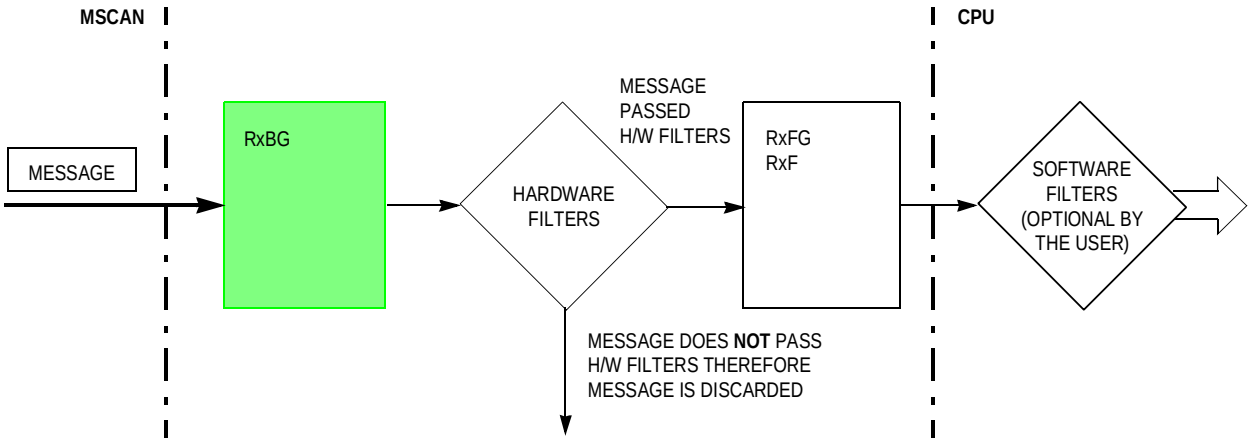


Figure 12. MSCAN's Receiver Structure

As shown in **Table 5**, each receive buffer consists of a 13-byte data structure:

- Four Identifier Registers
- Eight Data Segment Registers
- One Data Length Register

Table 5. Receive Buffer Data Structure

Register Name
Identifier Register 0
Identifier Register 1
Identifier Register 2
Identifier Register 3
Data Segment Register 0
Data Segment Register 1
Data Segment Register 2
Data Segment Register 3
Data Segment Register 4
Data Segment Register 5
Data Segment Register 6
Data Segment Register 7
Data Length Register

Identifier Registers

The Identifier Registers consists of either 11 bits (ID10–ID0) for the standard CAN 2.0 A/B format or 29 bits (ID28–ID0) for the extended CAN2.0 B format. See **Figure 13** and **Figure 14**.

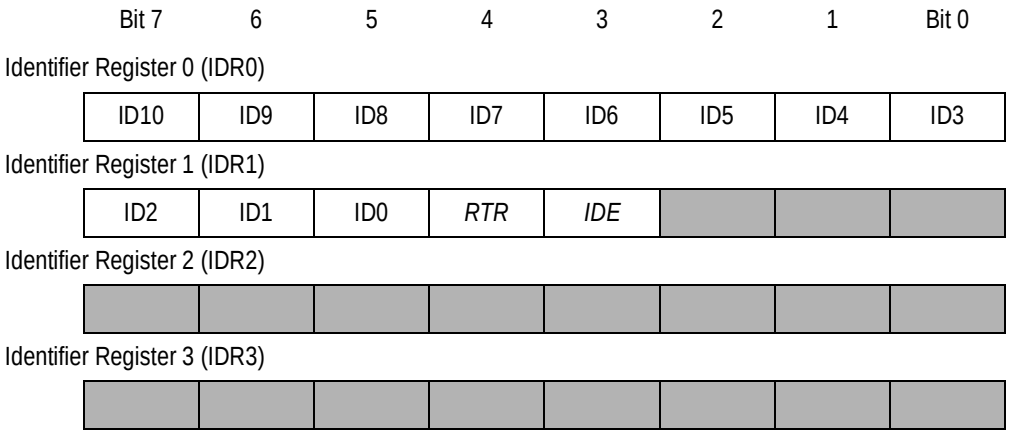


Figure 13. Standard Identifier Registers

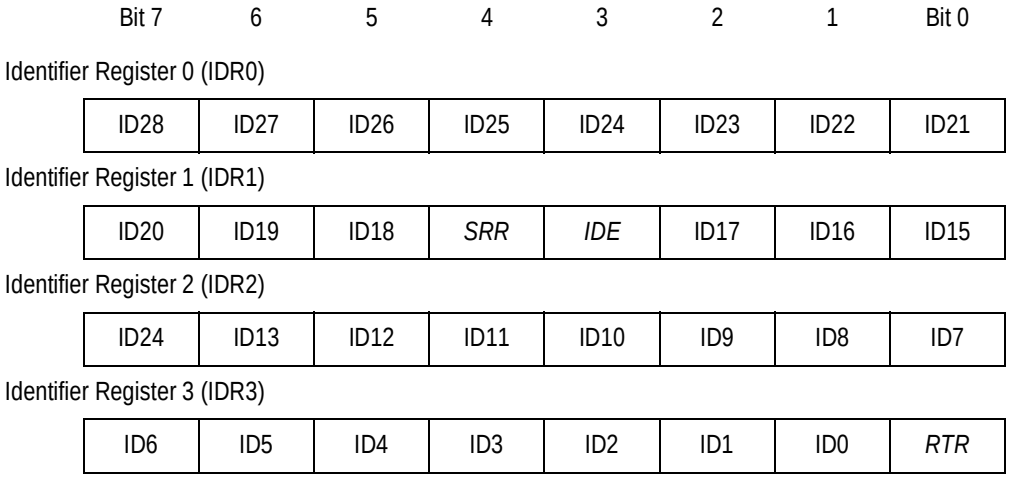


Figure 14. Extended Identifier Registers

SRR — Substitute Remote Request

This is a fixed recessive bit [recessive bit = 1 and dominant bit = 0] used only in the extended format mode. It must be set to 1 by the user for transmission buffers and will be stored as received on the CAN bus for receive buffers.

IDE — ID Extended

This flag indicates whether the extended or standard identifier format is applied in this buffer.

1 = Extended format (29 bit)
0 = Standard format (11 bit)

NOTE: *In the case of IDE = 0, ID17–ID0 and RTR bits are not applicable.*

RTR — Remote Transmission Request

This flag defines the type of message frame that will be transmitted onto the CAN bus and whether or not that frame contains data. In the case of a receive buffer, it indicates the status of the received frames and allows transmission support of a response data frame in software.

1 = Remote frame
0 = Data frame

Data Length Register

The Data Length Register contains information on the number of data bytes in the CAN message. The data byte count ranges from 0–8 bytes for the data frame (see **Table 6**).

Table 6. Data Length Code and Byte Count

Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

Data Segment Registers

The eight Data Segment Registers contain the data bytes to be received. The number of bytes being received is determined by the data length code in the Data Length Register.

Receive Process

The following steps describe the receive process.

1. The MSCAN finds a new data frame on the CAN bus and receives the bit stream into the Receive Background Buffer (RxBG) checking for errors (see **Error Overview**). If no errors are found, the MSCAN acknowledges the receipt of this data frame by sending an ACK symbol onto the CAN bus. If errors are detected, the message is discarded and the MSCAN waits for the next message to be received.

2. The hardware identifier acceptance filters are applied to the successfully received message in the RxBG. The Identifier of the message is compared with the hardware filters, composed of the acceptance and mask registers. The mask register specifies which particular bits to compare in the acceptance registers with the identifier of the message. For more details on hardware acceptance filters, refer to *Using The Freescale msCAN Filter Configuration Tool* (Freescale document order number AN2010/D).
3. When a message finds a filter match, the RxBG becomes the RxFG and is accessible by the CPU only if the Receive Buffer Full (RXF) bit in the CRFLG is clear (RXF = 0) indicating that the prior receive buffer (RxFG) was empty. The MSCAN Identifier Acceptance Control Register (CIDAC) is used to set the Identifier Acceptance Mode. CIDAC also indicates which identifier acceptance filter was hit by the message in the RxFG buffer that passed through the hardware filters. This information is imperative when performing additional filtering through software. If the message doesn't match one of the hardware filters then the message is discarded in the RxBG buffer and MSCAN can attempt to receive the next message into this buffer.

   The CIDAC registers for the MSCAN08 and MSCAN12 are shown in **Figure 15** and **Figure 16**. Note that they are different.

Bit 7	6	5	4	3	2	1	Bit 0
0	0	IDAM1	IDAM0	0	0	IDHIT1	IDHIT0

Figure 15. MSCAN08 Identifier Acceptance Control Register (CIDAC)

Bit 7	6	5	4	3	2	1	Bit 0
0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0

Figure 16. MSCAN12 Identifier Acceptance Control Register (CIDAC)

IDAM1–IDAM0 — Identifier Acceptance Mode

The CPU sets these flags to define the identifier acceptance filter organization. In the filter closed mode, no messages will be accepted so that the RxFG buffer will never be reloaded. See **Table 7** and **Table 8**.

IDHIT2–IDHIT0—Identifier Acceptance Hit Indicator

The MSCAN sets these flags to indicate an identifier acceptance hit. The IDHIT indicators are always related to the message in the RxFG buffer. When a message gets copied from RxBG to RxFG the indicators are updated as well. See **Table 9** and **Table 10**.

Table 7. MSCAN08 Identifier Acceptance Mode

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	One 32-bit acceptance filters
0	1	Two 16-bit acceptance filters
1	0	Four 8-bit acceptance filters
1	1	Filter closed

Table 8. MSCAN12 Identifier Acceptance Mode

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	Two 32-bit acceptance filters
0	1	Four 16-bit acceptance filters
1	0	Eight 8-bit acceptance filters
1	1	Filter closed

Table 9. MSCAN08 Identifier Acceptance Hit Indicators

IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	Filter 0 hit
0	1	Filter 1 hit
1	0	Filter 2 hit
1	1	Filter 3 hit

Table 10. MSCAN12 Identifier Acceptance Hit Indicators

IDHIT2	IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	0	Filter 0 hit
0	0	1	Filter 1 hit
0	1	0	Filter 2 hit
0	1	1	Filter 3 hit
1	0	0	Filter 4 hit
1	0	1	Filter 5 hit
1	1	0	Filter 6 hit
1	1	1	Filter 7 hit

4. After the RxBG is copied into the RxFG buffer, the RXF flag in the CRFLG register is set (RXF = 1). This receive buffer full event will result in a receive interrupt if the Receiver Full Interrupt Enable (RXFIE bit) of the MSCAN Receiver Interrupt Enable Register (CRIER) is enabled (RXFIE = 1). When the receive interrupt occurs, the user's receive handler has to read the received message from the RxFG and then clear the RXF flag to 0 in order to acknowledge the interrupt and to release the foreground buffer.
5. When both the RxBG and RxFG buffers contain correctly received and filtered messages, an overrun error occurs when a further message is being received from the bus. The new message will be ignored and an overrun error interrupt will occur if enabled. While in the overrun error condition, the MSCAN will stay synchronized to the CAN bus and is able to transmit messages but will ignore all incoming messages. To prevent the overrun condition from happening, the receive handler software must read the contents of the RxFG buffer and release it before the RxBG buffer is filled with the next filtered message that is ready for transfer to the RxFG buffer. For this reason, the receive interrupt servicing must be extremely efficient and any subsequent software filtering or other processing should be done outside the scope of the receive interrupt service routine.

Receive Interrupt

To enable a receive interrupt, the Receiver Full Interrupt Enable (RXFIE) bit should be set in the MSCAN Receiver Interrupt Enable Register (CRIER). The receive buffer full (successful message reception) event (RXF = 1 in the CRFLG (MSCAN Receiver Flag Register)) will trigger a receive interrupt. The receive interrupt handler would then copy the contents from the RxFG buffer into random-access memory (RAM) and apply, as necessary, secondary acceptance filters using software. Then, the receive interrupt handler would release the RxFG buffer by clearing the RXF flag in the CRFLG register. The RXF flag is cleared by writing a 1 to it while it is set.

The RXFIE bit is located in the CRIER register and the RXF bit is located in the CRFLG register (see **Figure 17** and **Figure 18**).

Bit 7	6	5	4	3	2	1	Bit 0
WUPIE	RWRNIE	TWRNIE	RERRIE	TERRIE	BOFFIE	OVRIE	RXFIE

Figure 17. MSCAN Receiver Interrupt Enable Register (CRIER)

RXFIE — Receiver Full Interrupt Enable

1 (set) = A receiver buffer full (successful message reception) event will result in a receive interrupt.

0 (clear) = No interrupt will be generated from this event.

Bit 7	6	5	4	3	2	1	Bit 0
WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF

Figure 18. MSCAN Receiver Flag Register (CRFLG)

RXF — Receive Buffer Full

The RXF flag is set by the MSCAN when a new message is available in the foreground receive buffer. This flag indicates whether the buffer is loaded with a correctly received message. After the CPU reads the message from the receive buffer, the RXF flag must be handshaken in order to release the buffer. When the RXF flag is set it prohibits the exchange of a successfully received message from the background receive buffer into the foreground buffer.

1 (set) = The receive buffer is full and a new message is available.
0 (clear) = The receive buffer is released (empty).

As discussed, a successful message received is one that has passed through the hardware filters set up by CIDAR and CIDMR. Additional filtering can be implemented through software. Maximize the hardware filtering process in order to improve the software filtering efficiency.

For more details on acceptance filters and secondary software filtering, refer to *Using The Freescale msCAN Filter Configuration Tool* (Freescale document order number AN2010/D).

A piece of sample code illustrating the receive interrupt follows.

NOTE: *The receive interrupt routine is short, just a couple steps. This is very important to achieve code/time efficiency.*

```
/***********************************************
        RECEIVE INTERRUPT
***********************************************/

@interrupt void receive_interrupt(void){
  volatile unsigned char received_length;
  volatile unsigned char count;

  Store_RX[m].ID0 = rx_buffer.ID0; // Stores the message's ID from the receive buffer
  Store_RX[m].Length =  rx_buffer.DLC; // Stores the message's length from the receive buffer
  Store_RX[m].Priority = rx_buffer.Priority // Stores the message's priority from the receive buffer

  received_length = Store_RX[m].Length; //Stores the Data Length of the message into a variable to be used in the next step

  // Store the data bytes of the message from the receive buffer
  // The number of data bytes in the message is determined by length
  for(count = 0; count < received_length; ++count){Store_RX[m].data[count] = rx_buffer.data[count];}

  HIT = CIDAC & 0x07; // Stores the filter heads

  CRFLG = 0x01;  //Clear the RXF flag in the CRFLG register to indicate this buffer is empty(released) and another
                 //message can be accepted into the receive buffer

  // DO NOT USE THE BSET Instruction Instead use the STA when clearing the RXF flag..
  }
```

Once again, the signals used in the CAN messages are stored in data structures which mirror the MSCAN receive buffers so that when storing data from the receive buffer all that is done is one for one data moves. In this case, the variable structure is called Store_RX. This format provides an efficient driver and improves execution time.

Error Overview

The MSCAN is designed to detect errors in transmission and reception of messages, as specified in the Bosch CAN specification version 2.0. Methods used to detect errors are:

- Bus Monitoring of what the MSCAN transmits
- Cyclic Redundancy Check (CRC)
- Message Frame Check

When a node detects an error it transmits an error flag onto the CAN bus, signaling all other nodes that a message error occurred. However, the host CPU is *NOT* signaled when a specific error has occurred. These messages, with error, are aborted and are retransmitted automatically.

The CAN nodes are able to distinguish between short disturbances and permanent failures by tracking errors over a period of time. A CAN node is able to determine whether the errors are due to faults of other nodes or itself and takes the appropriate steps to confine its operations on the bus. A CAN node can be in one of three states:

- Error Active
- Error Passive
- Bus Off

There are two counters that determine which state the node is in, the Transmit Error Counter (TEC) and the Receive Error Counter (REC). The TEC is associated with the node's transmit state machine and the REC is associated with the node's receive state machine.

NOTE: *Both error counters must only be read when in Sleep or Soft_RESET.*

In the Error Active state, both error counters are low so the node is fully able to participate in bus communication and is able to send Active Error Flags upon error detection at the node. The Error Passive state is entered when either error counter reaches a higher level and the node's communication on the bus becomes restricted, sending Passive Error Flags upon error detection. If the number of transmit errors of a node reaches an excessive level then the bus off state is entered and the node's communication on the bus is halted. The host CPU will be signaled only when one of these state changes occurs at the node.

Error Detection

The MSCAN is able to detect five types of errors. They are:

1. **Bit Error** — detected by the MSCAN's transmit state machine
2. **Stuff Error** — detected by the MSCAN's receiver state machine
3. **CRC Error** — detected by the MSCAN's receiver state machine
4. **Form Error** — detected by the MSCAN's receiver state machine
5. **Acknowledgement Error** — detected by the MSCAN's transmitter state machine

NOTE: *A node sending a bit on the bus also monitors the bus. That is, each node's transmit state machine compares the bit levels detected on the bus with the bit levels being transmitted.*

Bit Error

A node's transmitter detects a bit error if the bit value monitored is different from the bit value transmitted. An exception to this is when sending a recessive [1] bit and receiving a dominant [0] bit during the arbitration field or the ACK slot. Also, a node transmitting a PASSIVE Error Flag and receiving a dominant [0] bit doesn't interpret this as a Bit Error.

Stuff Error

Whenever five consecutive bits of equal value are transmitted, an extra bit of complementary value is automatically inserted into the bit stream. This provides edges for clock resynchronization. The receivers automatically de-stuff the extra bit of complementary value. So, when a receiver detects a 6^{th} consecutive equal level value (six dominant [0] bits or six recessive [1] bits) during a message field that should be encoded by bit stuffing a stuff error is detected.

CRC Error

A CRC (Cyclic Redundancy Check) error is detected by the node's receiver if the CRC calculated by the receiver is different from the CRC received in the message.

Form Error

The node's receiver detects a Form Error if a fixed form bit field contains one or more illegal bits. For example, a fixed form bit field would be the arbitration field which can be 12 (standard mode) / 32 (extended mode) bits in length; or the control field which has two reserved bits (r1 and r0) that need to be sent out as two dominants [0] in a row; or the CRC field which has the CRC Delimiter and needs to be a recessive [1] bit.

Acknowledgement Error

A node's transmitter detects an Acknowledge Error if it doesn't receive a dominant [0] bit during the ACK Slot. A dominant [0] bit in the ACK Slot indicates an acknowledgment occurred. The ACK Slot is in the Acknowledge field of the data frame. **Figure 19** provides a reminder of what the data frame consists of and shows what is inside the Acknowledge field.

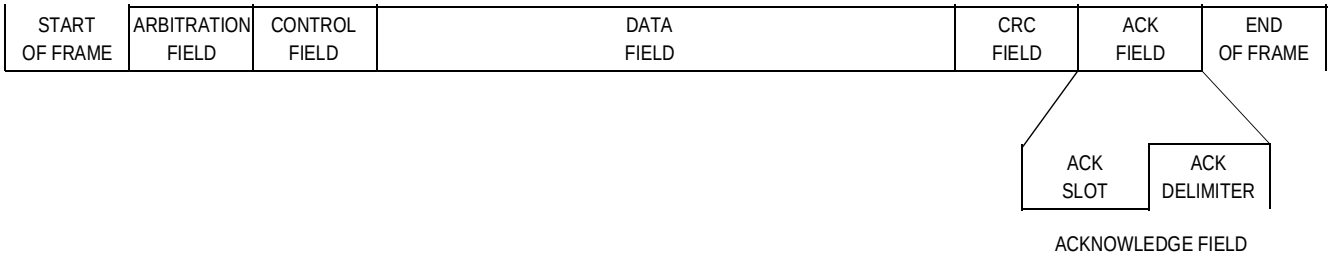


Figure 19. Data Frame

Error Signaling

When a node detects an error in transmission or reception, it signals the bus by transmitting an Error Flag. There are two kinds of error flags:

- Active Error Flag
- Passive Error Flag

These flags correspond to the error state of a node, which is governed by transmit and receive error counters. A node can be Error Active or Error Passive depending on these counters. When an Error Active Node detects an error condition, the node signals this by transmitting an Active Error Flag. When a Error Passive Node detects an error condition, the node signals this by transmitting a Passive Error Flag.

Active Error Flag

An Active Error Flag consists of six consecutive dominant [0] bits which:

- Overwrite other bits on the bus and break the rule of bit stuffing, or
- Destroy the ACK field or EOF field

When a node sends out an Active Error Flag after detecting an error, all the other nodes detect an error condition also and start to transmit their own Error Flags. So, a sequence of dominant [0] bits results from the overlaying of different error flag transmitted by individual nodes. As a result, the total length of the sequence varies between a minimum of 6 and a maximum of 12.

Passive Error Flag

The Passive Error Flag consists of six recessive [1] consecutive bits that do not overwrite other bits. A Passive Error Flag that is transmitted while any dominant bits are present on the bus will not be seen by any other nodes on the bus, including the transmitter.

Fault Confinement

Fault Confinement is a method for discriminating between temporary errors and permanent failures. Nodes that constantly generate errors are prevented from disturbing communication between error free nodes. To do this, each node has a Transmit Error Counter (TEC) and a Receive Error Counter (REC).

Both of these counters range from 0 to 255 and the hardware on the nodes update the count accordingly.

Each node can be in one of three error states:

Normal State/Initial State — TEC = REC = 0

- Error Active State
  A node is error active when TEC < 128 *AND* REC < 128.
  Sends an Active Error Flag when an error is detected.
- Error Passive State
  A node is passive error when 128 <= TEC <= 255 *OR* REC >= 128.
  Sends a Passive Error Flag when an error is detected. When a message is to be transmitted the node must wait for an additional amount of bus idle time (8 bit times) before transmission can begin.
- Bus Off State
  A node is bus off when TEC > 255.
  Not allowed to participate in bus communication (the output drivers are switched off), but permitted to become error active with both TEC and REC reset to 0 after 128 occurrences of 11 consecutive recessive [1] bits have been monitored on the bus. A minimum of 11 consecutive recessive [1] bits occur between messages (Acknowledgement Delimiter, End of Frame, and Intermission).

NOTE: *Receive errors cannot place the node into BUS OFF state. This keeps a lone node from going bus off due to lack of ACK symbols to attempted transmissions.*

Error Interrupts Overview

Error Interrupts are used to monitor the number of errors within a particular node and how they are affecting the communication between that node and other nodes in the system. The error counts are determined by the node's Transmit Error Counter (TEC) and Receive Error Counter (REC), which are controlled by the MSCAN. These counters dictate what states the nodes are in.

As shown in **Figure 20**, there are a total of four states a node can enter depending on the counters REC and TEC. The initial state is the No Warning/Error State when TEC and REC = 0. These states are:

1. No Warning/Error State
2. Receiver/Transmitter Warning State
3. Receiver/Transmitter Error Passive State
4. Bus Off State

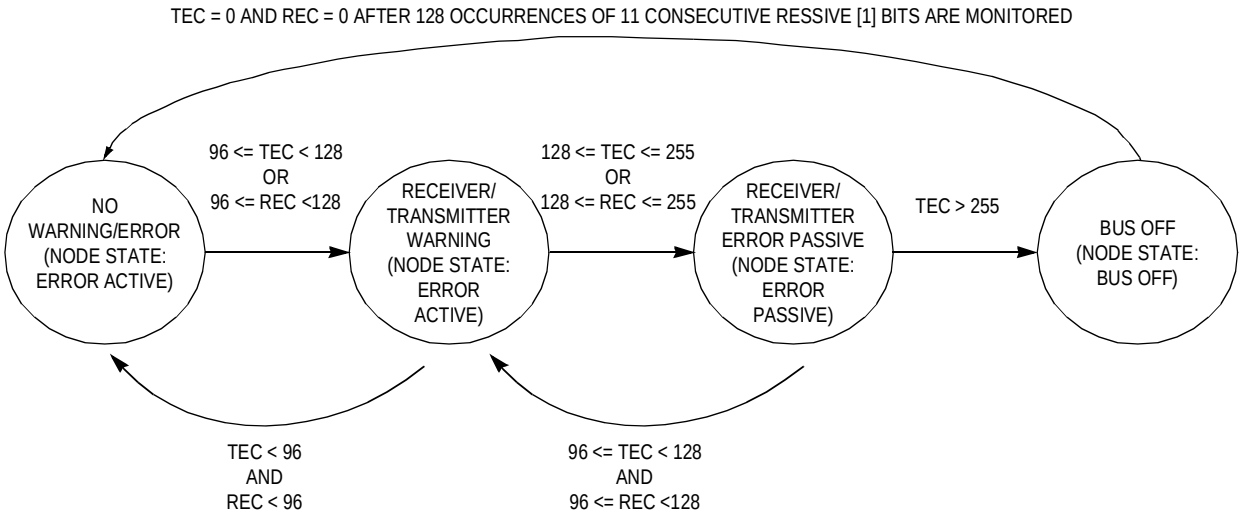


Figure 20. Node States

The node's receiver has three states depending on the value of the REC:

1. No Warning/Error State
2. Receiver Warning State
3. Receiver Error Passive State

The node's transmitter has four states depending on the value of the TEC:

1. No Warning State
2. Transmitter Warning State
3. Transmitter Error Passive State
4. Bus Off State

NOTE: *The Bus Off State is entered only when TEC > 255.*

Keeping track of error counts allows the user to decide whether further diagnostics or actions are required. The warnings and errors are directed to the CRFLG register (see **Figure 21**) where the flags will be set accordingly to notify that an error/warning has occurred.

Bit 7	6	5	4	3	2	1	Bit 0
WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF

Figure 21. MSCAN Receiver Flag Register (CRFLG)

There are a total of six sources of error interrupts:

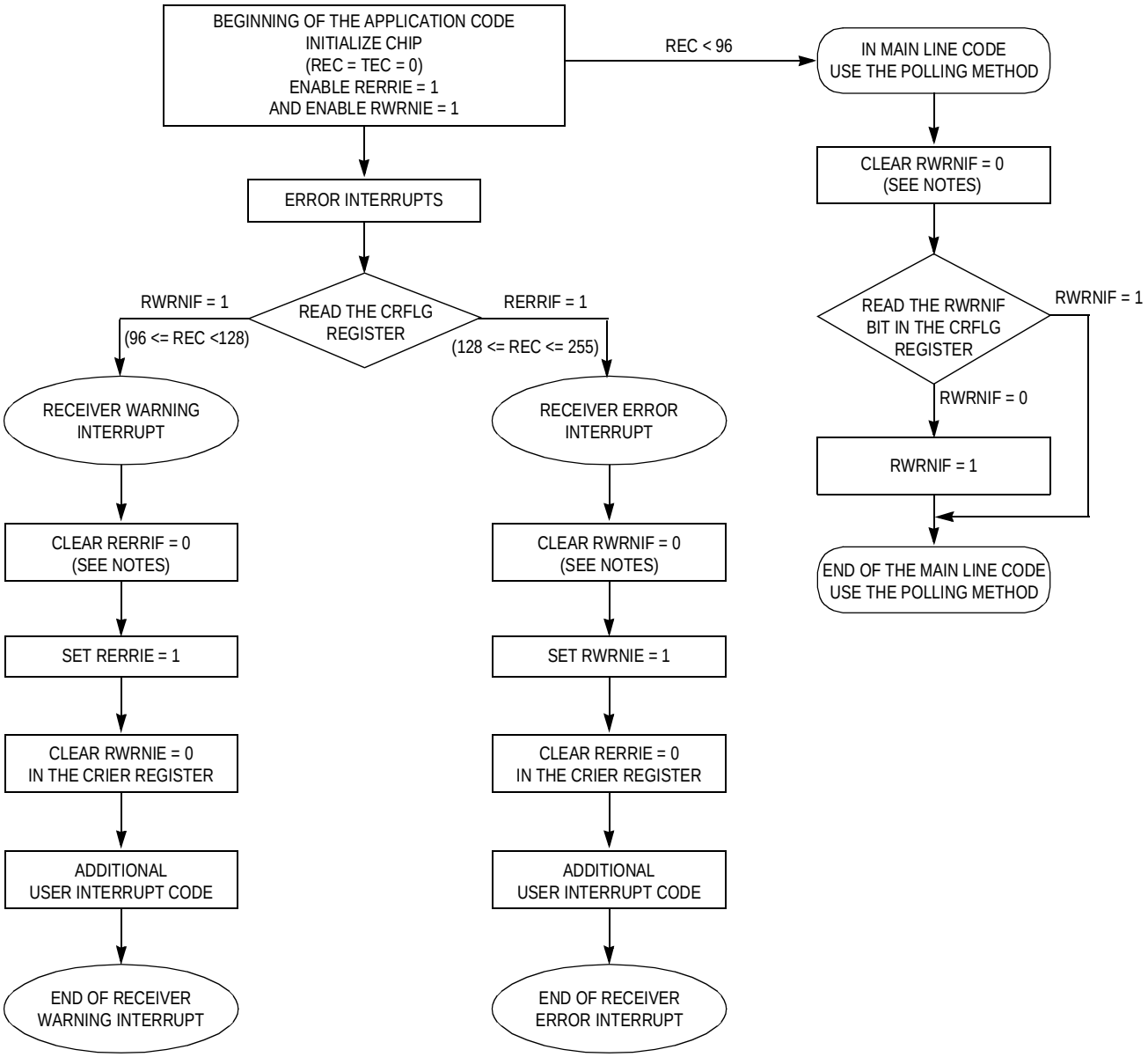
1. Receiver Warning Interrupt
2. Transmitter Warning Interrupt
3. Receiver Error Passive Interrupt
4. Transmitter Error Passive Interrupt
5. Bus Off Interrupt
6. Overrun Interrupt

There is only one Error ISR (interrupt service routine) to service the multiple sources of error interrupts. In the following subsections each of the six sources of error interrupts will be described in detail and exhibit the steps needed for the interrupt of each source. A sample piece of C code for the error ISR will be shown after the source descriptions.

Receiver Warning/Receiver Error Passive Interrupts

The example flowchart found in **Figure 22** illustrates generation of Receive Warning and Receive Error Passive Interrupts by the MSCAN based on the value of the REC count. The flowchart describes the minimal steps in the warning and error interrupt routines. Also shown is a separate routine where the polling method is utilized to make sure the interrupt sources wanted by the user are enabled.

As seen at the beginning of the flowchart the REC and TEC of the MSCAN are set to 0 out of reset. The user will determine which interrupts should be enabled based on their applications. In this case, both the Receiver Warning and Receiver Error Interrupts are enabled to show examples of both scenarios. The MSCAN adjusts the REC value when receive errors are detected and when messages are received successfully.



Notes:
1. In order to clear the bits in the CRFLG register write a 1 to the bit while it's set.
2. An exclusive OR operation 1 ⊕ 1 = 0 is performed on the bit.
3. Do not use the BSET instructions to clear bits. However, the LDA and STA instructions can be used. Take care that any C compiler being used does not use the BSET instructions.

Figure 22. Example Receiver Error Interrupt Flowchart

The REC dictates what state the node is going to be in according to the number of errors. Out of reset, the REC is initialized to 0 by the MSCAN. At this time, the node will be in the No Warning/Error (Error Active) State. When 96 <= REC < 128 the node is Error Active and in the Receiver Warning State. When 128 <= REC <= 255 the node is Error Passive and in the Receiver Error Passive State.

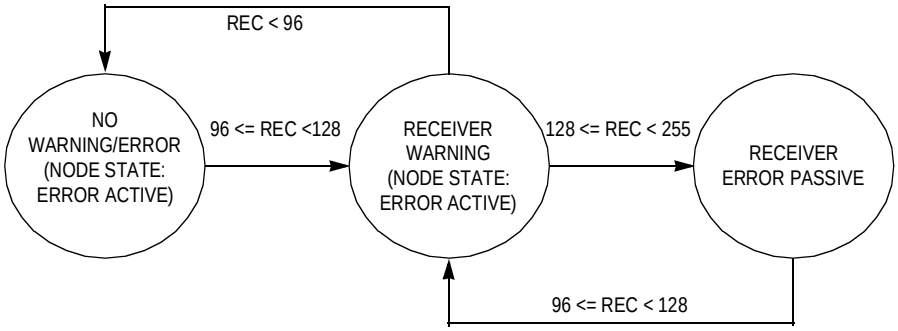


Figure 23. Receiver Error Polling Method

The rules REC follows to adjust its values are:

1. When a receiver detects an error, the REC is increased by one. Except when the detected error was a Bit Error during the sending of an ACTIVE Error Flag or an Overload which results in the REC being increased by eight.
2. When a receiver detects a dominant [0] bit as the first bit after sending an Error Flag, the REC is increased by eight.
3. When a node's receiver is Error Active and detects a Bit Error while sending an Active Error Flag or an Overload Flag, the REC is increased by eight.
4. A node can tolerate up to seven consecutive dominant [0] bits after sending an Error Flag (either Active or Passive). After detecting the 8^{th} consecutive dominant [0] bit and after each sequence of eight additional consecutive dominant [0] bits, every receiver increases its receive error count by eight [REC + 8].
5. After a message is received successfully (without error up to the ACK slot) and after successfully sending the ACK bit:
   - The Receive Error Count decreases by one if it was between 1 and 127 [REC – 1 if between 1 and 127]
   - Or, if the Receive Error Count was zero it stays at zero [REC = 0 no change]
   - Or, if the Receive Error Count was greater than 127, then it'll be set to a value between 119 and 127
     [If REC > 127, then 119 <= REC <= 127]

In summary, the REC increases by one for most RX errors, by eight on severe RX errors, and decrements by one on a successful RX.

NOTE: *More than one rule may apply during a given message transfer.*

Receiver Warning Interrupt

The Receiver Warning Interrupt tells the software driver that the node is in the error active state, and the REC value is in the range 96 <=< REC < 128. This implements a recommendation in the CAN specification which gives the application an indicator that there are issues on the bus before it reaches the point that communication is affected. Once REC crosses into this range, the RWRNIF (Receiver Warning Interrupt Flag) in CRFLG is set. To enable the Receiver Warning Error Interrupt, set the RWRNIE (Receiver Warning Interrupt Enable) bit in CRIER.

To handle the Receiver Warning Interrupt source:

1. Clear the Receiver Error Interrupt Flag (RERRIF) bit in by writing a 1 to it. (As mentioned in **Figure 22**, CRFLG utilizes the exclusive OR operation). The reason is that this state can be entered from the Receiver Error Passive State, where RERRIF was set. These interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
2. Next, set the Receiver Error Passive Interrupt Enable (RERRIE) bit in CRIER. The RERRIE bit is cleared in the Receiver Error Passive Interrupt routine when the MSCAN transitions to the Receiver Error Passive State.
3. Clear the RWRNIE bit in CRIER so that another Receiver Warning Interrupt will not execute repeatedly for the duration of this state (Receiver Warning State). This is due to the fact that interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
4. Any additional interrupt handling code.

Receiver Error Passive Interrupt Routine

The Receiver Error Passive Interrupt tells the software driver that the node is in the Error Passive State, and that the REC value is in the range 128 <=< REC <= 255, at which point RERRIF in CRFLG is set. To enable the Receiver Error Passive Interrupt Source, set RERRIE in CRIER.

To handle the Receiver Error Passive Interrupt source:

1. Clear the Receiver Warning Interrupt Flag (RWRNIF) bit in CRFLG by writing a 1 to it. (As mentioned in **Figure 23**, CRFLG utilizes the exclusive OR operation). The reason is that this state is entered from the Receiver Warning State, where RWRNIF was set. These interrupt flags remain set in a level sensitive manner as long as the setting condition remains.

2. Next, set the Receiver Warning Interrupt Enable (RWRNIE) bit in CRIER. The RWRNIE bit is cleared in the Receiver Warning Interrupt routine when the MSCAN transitions to the Receiver Warning State.
3. Clear the RERRIE bit in CRIER so that another Receiver Error Passive Interrupt will not execute repeatedly for the duration in this state (Receiver Error Passive State). This is due to the fact that interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
4. Any additional interrupt handling code

Separate Routine using the Polling Method

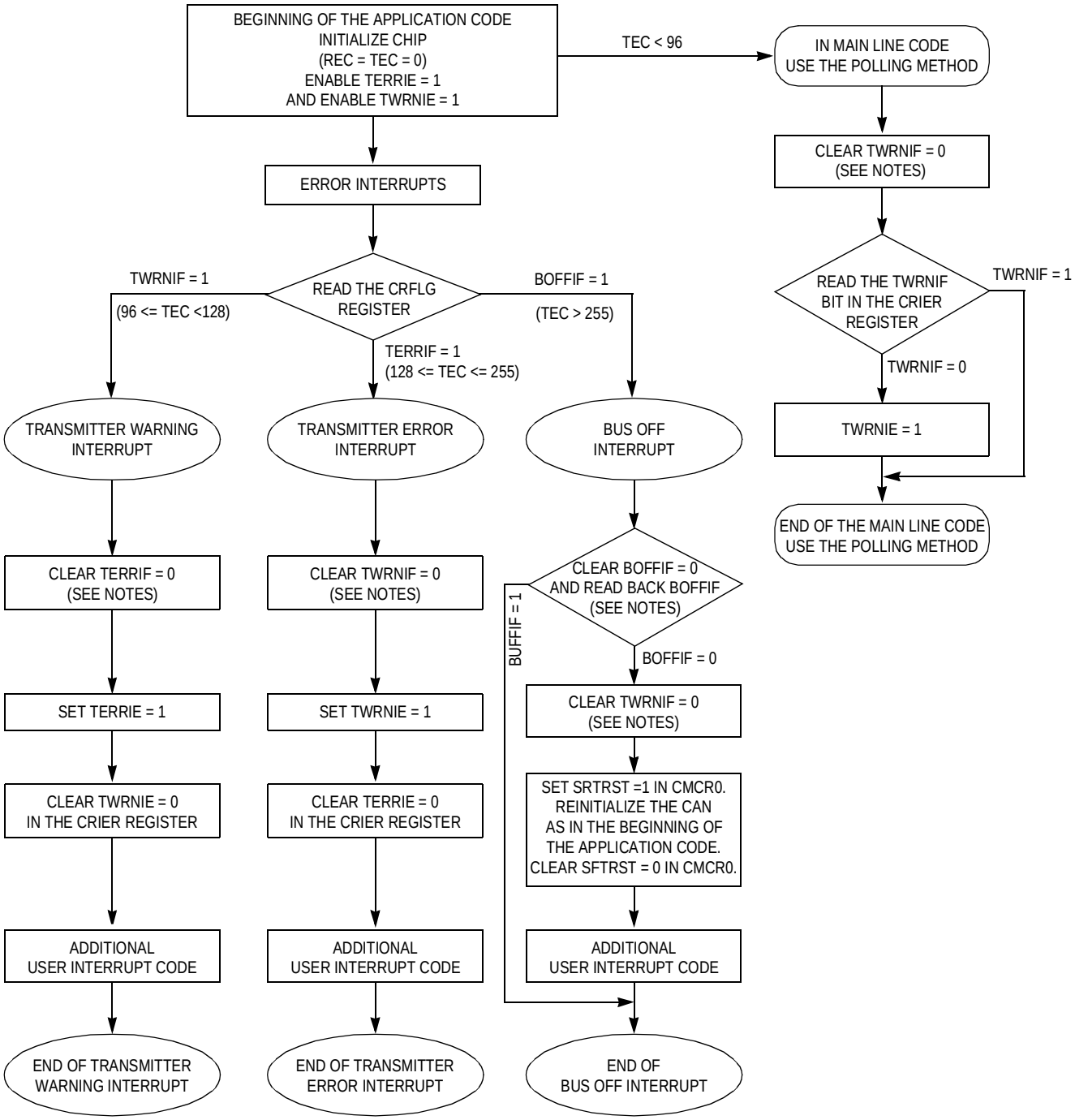
No interrupts are generated when the MSCAN goes from the receiver warning state to the No Error/Warning State where REC < 96. As seen from **Figure 23**, the RWRNIE bit of CRIER is disabled in the Receiver Warning Interrupt Routine. A separate polling routine can be used to determine when to re-enable the RWRNIE bit:

1. Clear the Receiver Warning Interrupt Flag (RWRNIF) by writing a 1 to it. (CRFLG utilizes the exclusive OR operation.)
2. Read the RWRNIF bit in CRFLG.
   a. If RWRNIF = 0, then REC < 96, so set the Receiver Warning Interrupt Enable (RWRNIE) bit in CRIER to re-enable the Receiver Warning Interrupt.
   b. If RWRNIF = 1, then the node is still in the Receiver Warning State (REC >= 96).

Transmitter Warning/Transmitter Error Passive Interrupts

Figure 24 is a flowchart example illustrating generation of Transmit Warning, Transmit Error Passive, and Bus Off Interrupts by the MSCAN based on the value of the TEC count. The flowchart describes the minimal steps in the interrupt service routines. Also shown is a separate routine where the polling method is utilized to make sure the interrupts wanted by the user are enabled.

As seen at the beginning of the flowchart the REC and TEC of the MSCAN are set to 0 out of reset. The user will determine which interrupts should be enabled based on their applications. In this case, the Transmitter Warning, Transmitter Error, and Bus Off Interrupts are enabled to show examples of all three scenarios. The MSCAN adjusts the TEC value when transmit errors are detected and when messages are transmitted successfully.



Notes:

1. In order to clear the bits in the CRFLG register write a 1 to the bit while it's set.
2. An exclusive OR operation 1 ⊕ 1 = 0 is performed on the bit.
3. Do not use the BSET instructions to clear bits. However, the LDA and STA instructions can be used. Take care that any C compiler being used does not use the BSET instructions.

Figure 24. Example Transmitter Error Interrupt Flowchart

The TEC dictates what state the node is in according to the number of transmit errors observed. Out of reset, the TEC is initialized to 0 by the MSCAN. At this time, the node will be in the No Warning/Error (Error Active) State. When 96 <= TEC < 128 then the node is Error Active and in the Transmitter Warning State. When 128 <= TEC <= 255 the node is error passive and in the Transmitter Error Passive State. When TEC > 255 the node enters the bus off state and does not transmit messages onto the bus until 128 occurrences of 11 consecutive recessive [1] bits are monitored on the bus. At which time, both the REC and TEC are reset to 0 thus entering the No Warning/Error State.

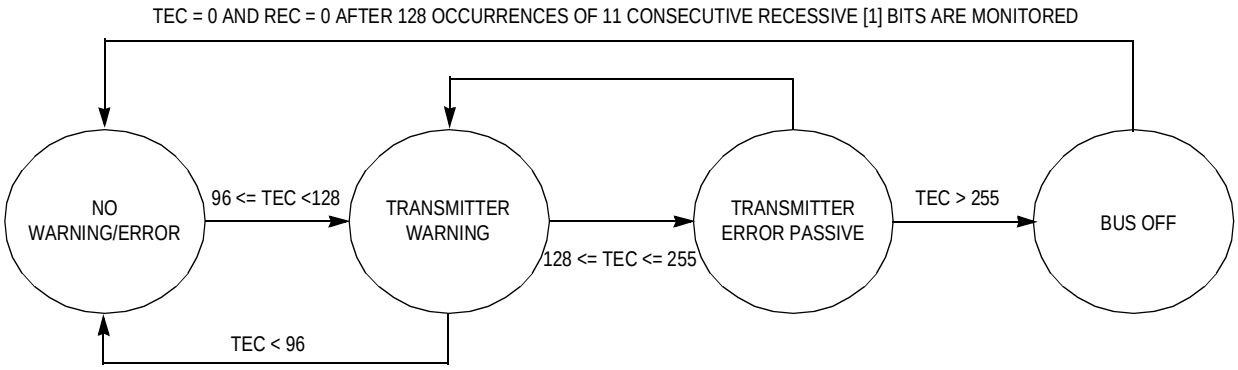


Figure 25. Transmitter Error Polling Method

The rules TEC follows to adjust its values are:

1. When a node detects a transmit error condition the transmitter sends an error flag to notify the MSCAN's bus and the Transmit Error Count is increased by eight.
2. When an Error Active node's transmitter detects a Bit Error while sending an Active Error Flag or an Overload Flag, the Transmit Error Count is increased by eight.
3. When an Error Active node detects an error condition it signals it by transmitting an Active Error Flag. As mentioned before the Active Error Flag consists of six consecutive dominant bits. The Error Flag's Form violates the Bit Stuffing rule(1). As a consequence, all other nodes detect an error condition and so each start to transmit an error flag. So, the sequence of dominant bits which actually can be monitored on the bus result from a superposition of different error flags transmitted by individual nodes. So, any node can tolerate up to seven consecutive dominant [0] bits after sending an error flag. After detecting the 8th

1. Bit Stuffing Rule: When five consecutive bits of equal value are transmitted, an extra bit of the opposite value is automatically inserted into the bit stream which provides edges for clock resynchronization. Then, the receivers automatically de-stuff this extra bit.

consecutive dominant [0] bit and after each sequence of eight additional consecutive dominant [0] bits, every transmitter increases its Transmit Error Count by eight.

4. After a message transmits successfully, by getting an ACK and no error until EOF is finished, the Transmit Error Count is decreased by one. Unless it is zero, where it will remain the same.
5. The TEC is unchanged when:
   – Transmitter is Error Passive
   AND
   Transmitter detects an Acknowledgement error because of not detecting a dominant [0] ACK
   AND
   Transmitter doesn't detect a dominant [0] bit while sending its Passive Error Flag
   – Transmitter sends an Error Flag because a Stuff Error occurred during arbitration whereby, the stuff bit is located before the RTR bit
   AND
   Stuff bit should have been recessive [1]
   AND
   Stuff bit has been sent as recessive [1] but is monitored as dominant [0].

In summary, the TEC increases by eight for all TX errors and decrements by one on a successful TX.

NOTE: *More than one rule may apply during a given message transfer.*

Transmitter Warning Interrupt

The Transmitter Warning Interrupt tells the software driver that the node is in the error active state, and the TEC value is in the range 96 <= REC < 128, at which point the TWRNIF (Transmitter Warning Interrupt Flag) in CRFLG is set. To enable the Transmitter Warning Error Interrupt, set the TWRNIE (Transmitter Warning Interrupt Enable) bit in CRIER.

To handle the Transmitter Warning Interrupt source:

1. Clear the Transmitter Error Interrupt Flag (TERRIF) bit in CRFLG by writing a 1 to it. (As mentioned in **Figure 24**, CRFLG utilizes the exclusive OR operation. The reason is that this state can be entered from the Transmitter Error Passive State, where the TERRIF was set.) These interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
2. Next, set the Transmitter Error Passive Interrupt Enable (TERRIE) bit in CRIER. The TERRIE bit is cleared in the transmitter error passive interrupt routine when the MSCAN transitions to the Transmitter Error Passive State.

3. Clear the Transmitter Warning Interrupt Enable (TWRNIE) bit in CRIER so that another Transmitter Warning Interrupt will not execute repeatedly for the duration of this state (transmitter warning state). This is due to the fact that interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
4. Any additional interrupt handling code.

Transmitter Error Passive Interrupt Routine

The Transmitter Error Passive Interrupt tells the software driver that the node is in the error passive state, and the TEC value is in the range 128 <= REC <= 255, at which point the TERRIF (Transmitter Error Passive Interrupt Flag) in CRFLG is set. To enable the Transmitter Error Passive Interrupt, set the TERRIE (Transmitter Error Passive Interrupt Enable) bit in CRIER.

To handle the Transmitter Error Passive Interrupt source:

1. Clear the Transmitter Warning Interrupt Flag (TWRNIF) bit in CRFLG by writing a 1 to it. (As mentioned in **Figure 24**, CRFLG utilizes the exclusive OR operation. The reason is that this state is entered from the transmitter warning state, where the TWRNIF was set.) These interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
2. Next, set the Transmitter Warning Interrupt Enable (TWRNIE) bit in CRIER. The TWRNIE bit is cleared in the Transmitter Warning Interrupt routine when the MSCAN transitions to the Transmitter Warning State.
3. Clear the Transmitter Error Passive Interrupt Enable (TERRIE) bit in CRIER so that another Transmitter Error Passive Interrupt will not execute repeatedly for the duration of this state (transmitter error passive state). This is due to the fact that interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
4. Any additional interrupt handling code.

Bus Off Interrupt

The bus off interrupt tells the software driver that the node is in the bus off state, and the TEC value is greater than 255. At this point the BOFFIF (bus off interrupt flag) in CRFLG is set. To enable the Bus Off Interrupt, set the BOFFIE (Bus Off Interrupt Enable) bit in CRIER. When the node is in the Bus Off State it is not allowed to participate in bus communication because it has generated too many transmit errors; therefore, disturbing communication between itself and other nodes. After 128 occurrences of 11 consecutive recessive [1] bits have been monitored on the bus the REC and TEC are reset to 0, and the node returns to the error active state (No Warning/Error State).

To handle the Bus Off Interrupt source:

1. Clear the BOFFIF bit in CRFLG by writing a 1 to it. (As mentioned in **Figure 24**, CRFLG utilizes the exclusive OR operation.) Then read back this bit.
   – If BOFFIF = 1, jump out of this interrupt routine
   – If BOFFIF = 0, this signifies that the node has monitored 128 occurrences of 11 consecutive recessive [1] bits at which point the node's REC and TEC are reset to 0 and out of the bus off state. So continue on to step 2.
2. Clear the Transmitter Error Interrupt Flag (TERRIF) bit in CRFLG by writing a 1 to it. (As mentioned in **Figure 24**, CRFLG utilizes the exclusive OR operation. The reason is that this state is entered from the Transmitter Error Passive State, where TERRIF was set.) These interrupt flags remain set in a level sensitive manner as long as the setting condition remains.
3. Set the Soft Reset bit (soft reset state) in the Control Register 0 (CMCR0). *The registers CMCR1, CBTR0, CBTR1, CIDAC, CIDAR0–CIDAR7, CIDMR0–CIDMR7 can only be written by the CPU when the MSCAN12 is in Soft Reset State.*
4. Reinitialize the MSCAN module as in the beginning of the application code to make sure none of the control registers have been corrupted.
5. Clear the Soft Reset bit (normal operation) in CMCR0.
6. Any additional interrupt code.

Keep in mind not all system requirements allow recovery. Some system requirements want to remain in the bus off state. Therefore, those other system requirements will have different flows compared to the example given here. Alternate system requirements may have other transmit requests. When a node is in bus off state it does not transmit or receive any messages. It is only counting the recessive bits and when 128 occurrences of 11 recessive bits are monitored the node is out of the bus off state and both the TEC and REC are reset to 0 and go into normal operation. The systems limited operating strategy (LOS) capabilities are determined by the network architecture and this node's (in the Bus Off State) operating requirements.

Separate Routine Using the Polling Method

No interrupts are generated when the MSCAN goes from the transmitter warning state to the no error/warning state where TEC < 96. As shown in **Figure 24**, the TWRNIE bit of CRIER is disabled in the Transmitter Warning Interrupt Routine.

A separate polling routine can be used to determine when to re-enable the TWRNIE bit.

1. Clear the Transmitter Warning Interrupt Flag (TWRNIF) by writing a 1 to it. (CRFLG utilizes the exclusive OR operation.)
2. Read the TWRNIF bit in CRFLG.
   a. If TWRNIF = 0, then REC < 96, so set the Transmitter Warning Interrupt Enable (TWRNIE) bit in CRIER to re-enable the Transmitter Warning Interrupt.
   b. If TWRNIF = 1, then the node is still in the Transmitter Warning State (REC >= 96).

An example of the error interrupt ISR in C-code from the five of the six sources described above follows.

NOTE: *The error routine is contained in only one ISR code. As seen in the example below, each of the interrupt sources is contained within one ISR.*

```
@interrupt void Error_Interrupt(void){

if(crflg.RWRNIF == 1){
        if(crflg.RERRIF == 1){ CRFLG = 0x10;}//crflg.RERRIF = 1;}  // Clear the RERRIF Flag in the CRFLG register
        crier.RERRIE = 1;  // Set the RERRIE bit in the CRIER register to enable Receiver Error Interrupt
        crier.RWRNIE = 0;  // Clear the RWRNIE bit in the CRIER register
        rx_warning = rx_warning + 1; // An example of additional code to keep track of receiver warning errors
        }

if(crflg.RERRIF == 1){
        if(crflg.RWRNIF == 1){CRFLG = 0x40;}//crflg.RWRNIF = 1;} // Clear the RWRNIF Flag in the CRFLG register
        crier.RWRNIE = 1; // Set the RWRNIE bit in the CRIER register to enable Receiver Warning Interrupts
        crier.RERRIE = 0; // Clear the RERRIE bit in the CRIER register
        rx_error = rx_error + 1; // An example of additional code to keep track of receiver errors
        }

if(crflg.TWRNIF == 1){
        if(crflg.TERRIF == 1){CRFLG = 0x08;}//crflg.TERRIF = 1;} // Clear the TERRIF Flag in the CRFLG register
        crier.TERRIE = 1; // Set the TERRIE bit in the CRIER register to enable the Transmitter Error Interrupt
        crier.TWRNIE = 0; // Clear the TWRNIE bit in the CRIER register
        tx_warning  = tx_warning + 1; // An example of additional code to keep track of transmitter warning errors
        }

if(crflg.TERRIF == 1){
        if(crflg.TWRNIF == 1){CRFLG = 0x20;}//crflg.TWRNIF = 1;} // Clear the TWRNIF Flag in the CRFLG register
        crier.TWRNIE = 1; // Set the TWRNIE bit in the CRIER register to enable Transmitter Warning Interrupts
        crier.TERRIE = 0; // Clear the TERRIE bit in the CRIER register
        tx_error = tx_error + 1; // An example of additional code to keep track of transmitter errors
        if(crflg.BOFFIF == 1){CRFLG = 0x04;}
        }

error_value3 = CTXERR;

if(crflg.BOFFIF == 1){
        crflg.BOFFIF = 1;  // Try to clear the BOFFIF Flag in the CRFLG register
        if(BOFFIF == 0){
                if(crflg.TERRIF == 1){CRFLG = 0x08;}//crflg.TERRIF = 1;} // Clear the TERRIF Flag in the CRFLG register
                cmcr0.SFTRES = 1; // Set the Soft Reset to reinitialize the CAN like in the beginning of the application
                        initialize(); //Reinitialize the MSCAN12 as in the beginning of the code
                cmcr0.SFTRES = 0;
                } //End of If Statement crflg.BOFFIF == 0
        }

} // End of Error Interrupt
```

Overrun Interrupt

An Overrun Interrupt occurs when a new incoming message appears on the CAN bus when both the RxBG and RxFG buffers contain correctly received and filtered messages. To enable the Overrun Interrupt, set the OVRIE (Overrun Interrupt Enable) bit in CRIER which will result in an Overrun Interrupt when the OVRIF flag sets in CRFLG. The new message and all further incoming messages will be ignored until the Foreground Receive Buffer (RxFG) is released. The MSCAN will stay synchronized to the CAN bus and is able to transmit messages. To prevent the Overrun Condition, the receive handler software must read the contents of the RxFG buffer and release it before the RxBG buffer is filled with the next filtered message.

Summary

This application note covers the interrupts and their operation on the MSCAN08 (for the HC08 Family) and MSCAN12 (for the HC12 Family). The example interrupt service routine flowcharts (**Figure 22** and **Figure 24**) show the minimal steps necessary to service each interrupt, but user's actual interrupt routines will vary depending on system requirements. Please refer to the relevant MSCAN and microcontroller unit (MCU) manuals for a complete description of the registers and other non-interrupt details of specific devices (see **References**).

References

1. *CAN Bosch Controller Area Network (CAN) Version 2.0 Protocol Standard* (order number BCANPSV2.0/D)
2. *ISO 11898* — can be found on the World Wide Web at: http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=20380&ICS1=43&ICS2=40&ICS3=15
3. *M68HC12B Family Advance Information* (Freescale document order number M68HC12B/D)
4. *68HC912D60/68HC12D60 Advance Information* (Freescale document order number MC68HC912D60/D)
5. *MC68HC912D60A Technical Data* (Freescale document order number MC68HC912D60A/D)
6. *MC68HC912DT128A/MC68HC912DG128A Technical Data* (Freescale document order number MC68HC912DT128A/D)
7. *MC68HC08AZ32 Advance Information* (Freescale document order number MC68HC08AZ32/D)
8. *MC68HC08AZ32A Advance Information* (Freescale document order number MC68HC08AZ32A/D)

9. *MC68HC08AZ60 Advance Information* (Freescale document order number MC68HC08AZ60/D)
10. *MC68HC908AZ60 Technical Data* (Freescale document order number MC68HC908AZ60/D)
11. *MC68HC908AZ60A Advance Information* (Freescale document order number MC68HC908AZ60A/D)
12. *Using the Freescale MSCAN Filter Configuration Tool* (Freescale document order number AN2010/D)
13. *MSCAN Low-Power Applications* (Freescale document order number AN2255/D)

[redacted]

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com