Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of "Quality Parts,Customers Priority,Honest Operation,and Considerate Service",our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

MCP2510

Stand-Alone CAN Controller with SPI™ Interface

Features

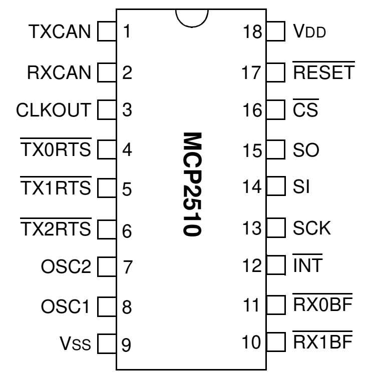
- Implements Full CAN V2.0A and V2.0B at 1 Mb/s:
  - 0 - 8 byte message length
  - Standard and extended data frames
  - Programmable bit rate up to 1 Mb/s
  - Support for remote frames
  - Two receive buffers with prioritized message storage
  - Six full acceptance filters
  - Two full acceptance filter masks
  - Three transmit buffers with prioritization and abort features
  - Loop-back mode for self test operation
- Hardware Features:
  - High Speed SPI Interface (5 MHz at 4.5V I temp)
  - Supports SPI modes 0,0 and 1,1
  - Clock out pin with programmable prescaler
  - Interrupt output pin with selectable enables
  - 'Buffer full' output pins configureable as interrupt pins for each receive buffer or as general purpose digital outputs
  - 'Request to Send' input pins configureable as control pins to request immediate message transmission for each transmit buffer or as general purpose digital inputs
  - Low Power Sleep mode
- Low power CMOS technology:
  - Operates from 3.0V to 5.5V
  - 5 mA active current typical
  - 10 µA standby current typical at 5.5V
- 18-pin PDIP/SOIC and 20-pin TSSOP packages
- Temperature ranges supported:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

Description

The Microchip Technology Inc. MCP2510 is a Full Controller Area Network (CAN) protocol controller implementing CAN specification V2.0 A/B. It supports CAN 1.2, CAN 2.0A, CAN 2.0B Passive, and CAN 2.0B Active versions of the protocol, and is capable of transmitting and receiving standard and extended messages. It is also capable of both acceptance filtering and message management. It includes three transmit buffers and two receive buffers that reduce the amount of microcontroller (MCU) management required. The MCU communication is implemented via an industry standard Serial Peripheral Interface (SPI) with data rates up to 5 Mb/s.

Package Types

18 LEAD PDIP/SOIC



20 LEAD TSSOP

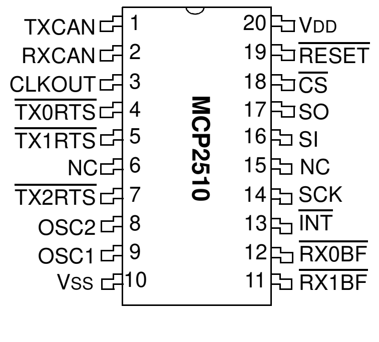


Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)

When contacting a sales office, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com** to receive the most current information on all of our products.

© 2007 Microchip Technology Inc.

1.0 DEVICE FUNCTIONALITY

1.1 Overview

The MCP2510 is a stand-alone CAN controller developed to simplify applications that require interfacing with a CAN bus. A simple block diagram of the MCP2510 is shown in Figure 1-1. The device consists of three main blocks:

1. The CAN protocol engine.
2. The control logic and SRAM registers that are used to configure the device and its operation.
3. The SPI protocol block.

A typical system implementation using the device is shown in Figure 1-2.

The CAN protocol engine handles all functions for receiving and transmitting messages on the bus. Messages are transmitted by first loading the appropriate message buffer and control registers. Transmission is initiated by using control register bits, via the SPI interface, or by using the transmit enable pins. 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 the user defined filters to see if it should be moved into one of the two receive buffers.

The MCU interfaces to the device via the SPI interface. Writing to and reading from all registers is done using standard SPI read and write commands.

Interrupt pins are provided to allow greater system flexibility. There is one multi-purpose interrupt pin as well as specific interrupt pins for each of the receive registers that can be used to indicate when a valid message has been received and loaded into one of the receive buffers. Use of the specific interrupt pins is optional, and the general purpose interrupt pin as well as status registers (accessed via the SPI interface) can also be used to determine when a valid message has been received.

There are also three pins available to initiate immediate transmission of a message that has been loaded into one of the three transmit registers. Use of these pins is optional and initiating message transmission can also be done by utilizing control registers accessed via the SPI interface.

Table 1-1 gives a complete list of all of the pins on the MCP2510.

FIGURE 1-1: BLOCK DIAGRAM

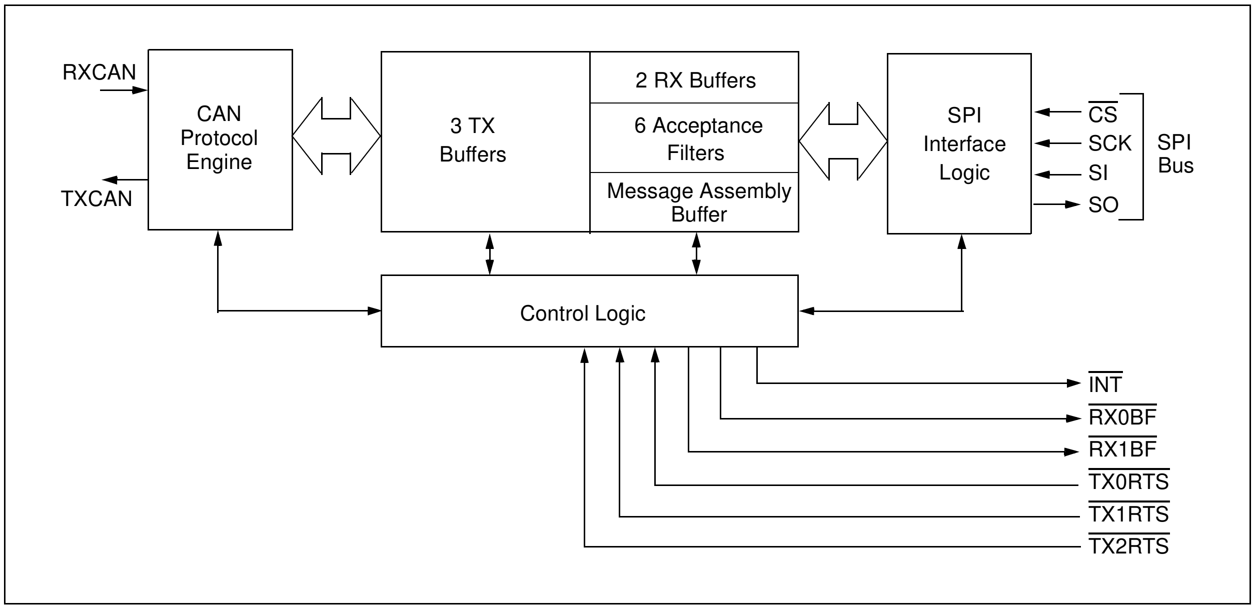


FIGURE 1-2: TYPICAL SYSTEM IMPLEMENTATION

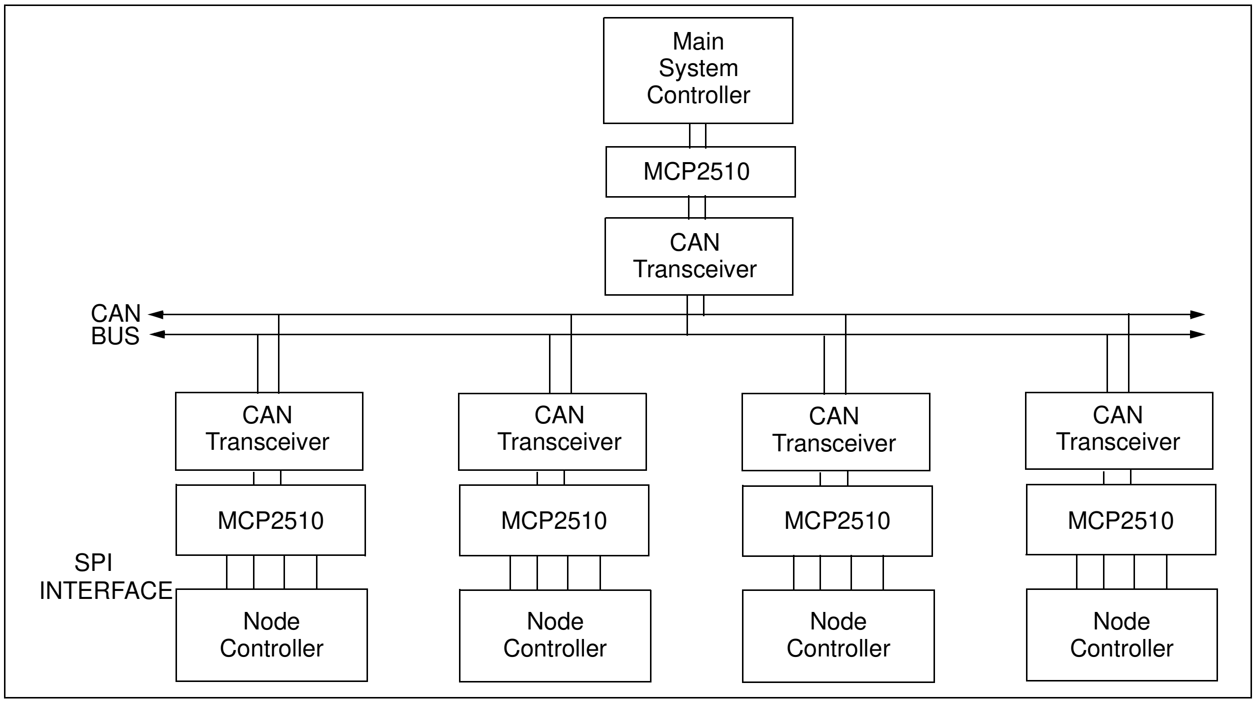


TABLE 1-1: PIN DESCRIPTIONS

Name	DIP/ SOIC Pin #	TSSOP Pin #	I/O/P Type	Description
TXCAN	1	1	O	Transmit output pin to CAN bus
RXCAN	2	2	I	Receive input pin from CAN bus
CLKOUT	3	3	O	Clock output pin with programmable prescaler
$\overline{\text{TX0RTS}}$	4	4	I	Transmit buffer TXB0 request to send or general purpose digital input. 100 kΩ internal pullup to V_{DD}
$\overline{\text{TX1RTS}}$	5	5	I	Transmit buffer TXB1 request to send or general purpose digital input. 100 kΩ internal pullup to V_{DD}
$\overline{\text{TX2RTS}}$	6	7	I	Transmit buffer TXB2 request to send or general purpose digital input. 100 kΩ internal pullup to V_{DD}
OSC2	7	8	O	Oscillator output
OSC1	8	9	I	Oscillator input
V_{SS}	9	10	P	Ground reference for logic and I/O pins
$\overline{\text{RX1BF}}$	10	11	O	Receive buffer RXB1 interrupt pin or general purpose digital output
$\overline{\text{RX0BF}}$	11	12	O	Receive buffer RXB0 interrupt pin or general purpose digital output
$\overline{\text{INT}}$	12	13	O	Interrupt output pin
SCK	13	14	I	Clock input pin for SPI interface
SI	14	16	I	Data input pin for SPI interface
SO	15	17	O	Data output pin for SPI interface
$\overline{\text{CS}}$	16	18	I	Chip select input pin for SPI interface
$\overline{\text{RESET}}$	17	19	I	Active low device reset input
V_{DD}	18	20	P	Positive supply for logic and I/O pins
NC	—	6,15	—	No internal connection

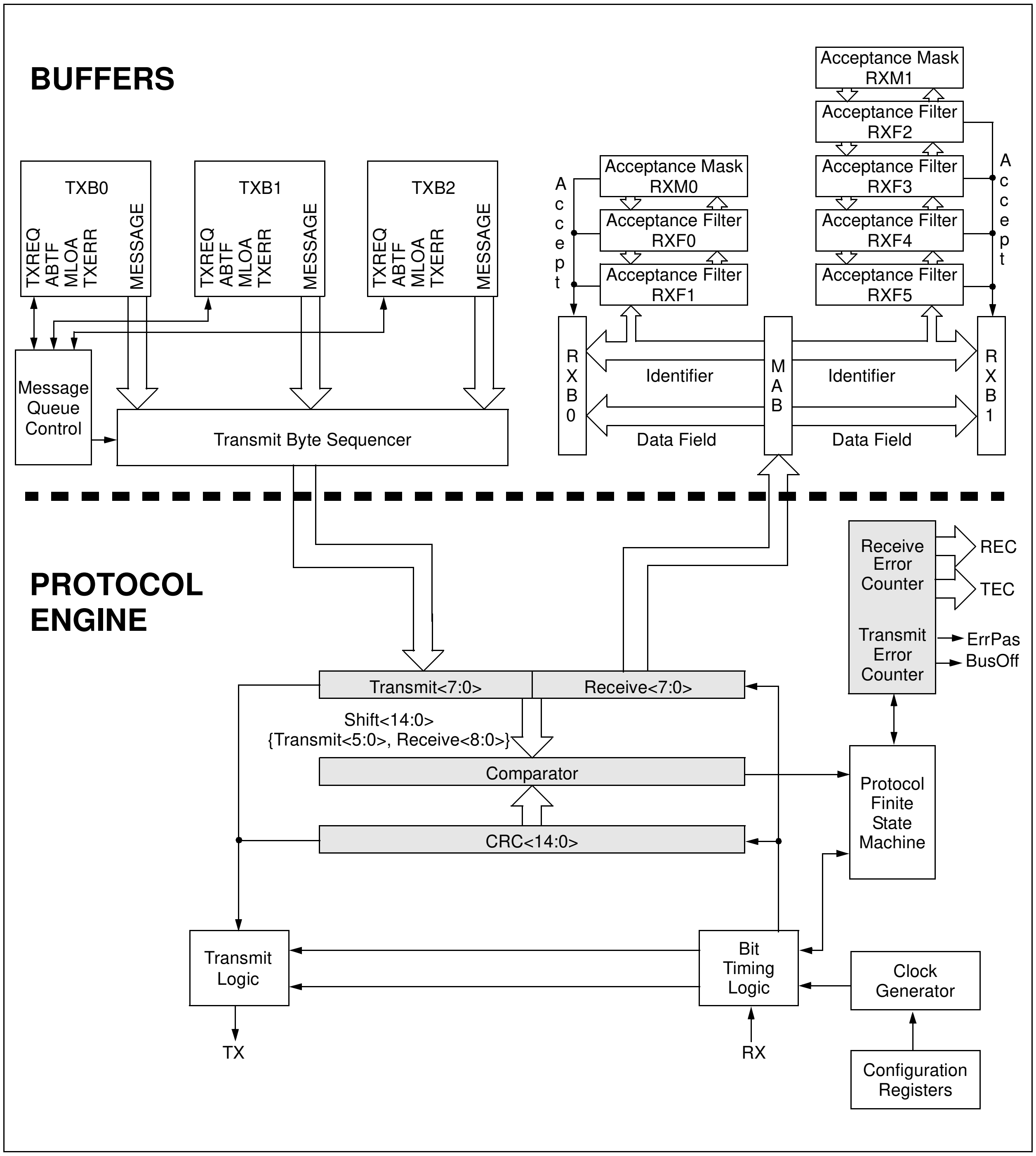
Note: Type Identification: I=Input; O=Output; P=Power

© 2007 Microchip Technology Inc.

1.2 Transmit/Receive Buffers

The MCP2510 has three transmit and two receive buffers, two acceptance masks (one for each receive buffer), and a total of six acceptance filters. Figure 1-3 is a block diagram of these buffers and their connection to the protocol engine.

FIGURE 1-3: CAN BUFFERS AND PROTOCOL ENGINE BLOCK DIAGRAM



1.3 CAN Protocol Engine

The CAN protocol engine combines several functional blocks, shown in Figure 1-4. These blocks and their functions are described below.

1.4 Protocol Finite State Machine

The heart of the engine is the Finite State Machine (FSM). This state machine sequences through messages on a bit by bit basis, changing states as the fields of the various frame types are transmitted or received. The FSM is a sequencer controlling the sequential data stream between the TX/RX Shift Register, the CRC Register, and the bus line. The FSM also controls the Error Management Logic (EML) and the parallel data stream between the TX/RX Shift Registers and the buffers. The FSM insures that the processes of reception, arbitration, transmission, and error signaling are performed according to the CAN protocol. The automatic retransmission of messages on the bus line is also handled by the FSM.

1.5 Cyclic Redundancy Check

The Cyclic Redundancy Check Register generates the Cyclic Redundancy Check (CRC) code which is transmitted after either the Control Field (for messages with 0 data bytes) or the Data Field, and is used to check the CRC field of incoming messages.

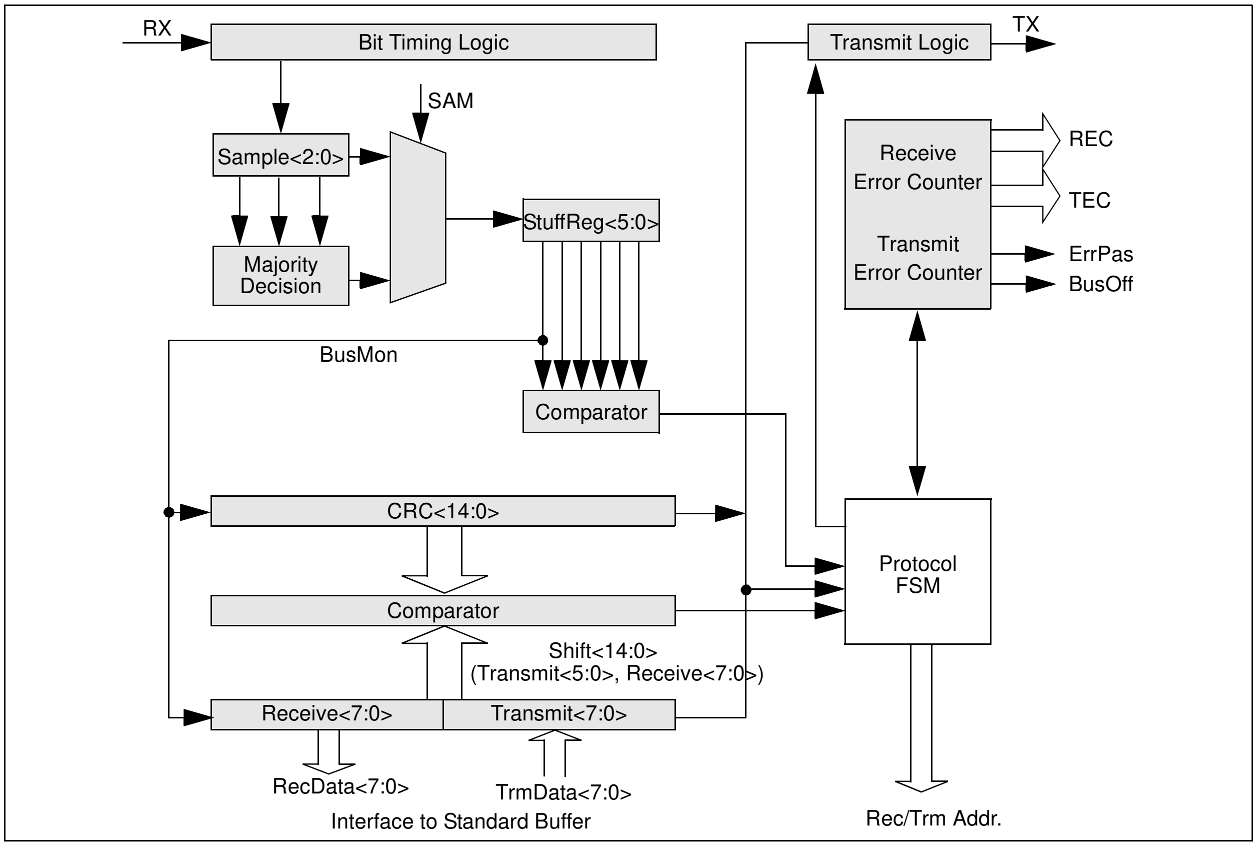
1.6 Error Management Logic

The Error Management Logic is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter (REC) and the Transmit Error Counter (TEC), are incremented and decremented by commands from the Bit Stream Processor. According to the values of the error counters, the CAN controller is set into the states error-active, error-passive or bus-off.

1.7 Bit Timing Logic

The Bit Timing Logic (BTL) monitors the bus line input and handles the bus related bit timing according to the CAN protocol. The BTL synchronizes on a recessive to dominant bus transition at Start of Frame (hard synchronization) and on any further recessive to dominant bus line transition if the CAN controller itself does not transmit a dominant bit (resynchronization). The BTL also provides programmable time segments to compensate for the propagation delay time, phase shifts, and to define the position of the Sample Point within the bit time. The programming of the BTL depends upon the baud rate and external physical delay times.

FIGURE 1-4: CAN PROTOCOL ENGINE BLOCK DIAGRAM



© 2007 Microchip Technology Inc.

2.0 CAN MESSAGE FRAMES

The MCP2510 supports Standard Data Frames, Extended Data Frames, and Remote Frames (Standard and Extended) as defined in the CAN 2.0B specification.

2.1 Standard Data Frame

The CAN Standard Data Frame is shown in Figure 2-1. In common with all other frames, the frame begins with a Start Of Frame (SOF) bit, which is of the dominant state, which allows hard synchronization of all nodes.

The SOF is followed by the arbitration field, consisting of 12 bits; the 11-bit Identifier and the Remote Transmission Request (RTR) bit. The RTR bit is used to distinguish a data frame (RTR bit dominant) from a remote frame (RTR bit recessive).

Following the arbitration field is the control field, consisting of six bits. The first bit of this field is the Identifier Extension (IDE) bit which must be dominant to specify a standard frame. The following bit, Reserved Bit Zero (RB0), is reserved and is defined to be a dominant bit by the can protocol. the remaining four bits of the control field are the Data Length Code (DLC) which specifies the number of bytes of data contained in the message.

After the control field is the data field, which contains any data bytes that are being sent, and is of the length defined by the DLC above (0-8 bytes).

The Cyclic Redundancy Check (CRC) Field follows the data field and is used to detect transmission errors. The CRC Field consists of a 15-bit CRC sequence, followed by the recessive CRC Delimiter bit.

The final field is the two-bit 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 field and may not be overwritten by a dominant bit.

2.2 Extended Data Frame

In the Extended CAN Data Frame, the SOF bit is followed by the arbitration field which consists of 32 bits, as shown in Figure 2-2. The first 11 bits are the most significant bits (Base-ID) of the 29-bit identifier. These 11 bits are followed by the Substitute Remote Request (SRR) bit which is defined to be recessive. The SRR bit is followed by the IDE bit which is recessive to denote an extended CAN frame.

It should be noted that if arbitration remains unresolved after transmission of the first 11 bits of the identifier, and one of the nodes involved in the 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 (Extended ID) and the 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 IDE bit can remain at the same bit position in both standard and extended frames.

Following the arbitration field is the six-bit control field. the first two bits of this field are reserved and must be dominant. the remaining four bits of the control field are the Data Length Code (DLC) which specifies the number of data bytes contained in the message.

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 (see Section 2.1).

2.3 Remote Frame

Normally, data transmission is performed on an autonomous basis by the data source node (e.g. a sensor sending out a data frame). It is possible, however, for a destination node to request data from the source. To accomplish this, 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 in response to the remote frame request.

There are two differences between a remote frame (shown in Figure 2-3) and a data frame. First, the RTR bit is at the recessive state, and second, there is no data field. In the 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.

2.4 Error Frame

An Error Frame is generated by any node that detects a bus error. An error frame, shown in Figure 2-4, consists of two fields, an error flag field followed by an error delimiter field. There are two types of error flag fields. Which type of error flag field is sent depends upon the error status of the node that detects and generates the error flag field.

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 returns 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 by the error delimiter field. The error-passive flag consists of six consecutive recessive bits, and the error frame for an error-passive node consists of 14 recessive bits. From this, it follows that unless the bus error is detected by the node 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 transmitting node generates an error-passive flag then this will 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 six consecutive recessive bits on the bus before attempting to rejoin bus communications.

The error delimiter consists of eight recessive bits and allows the bus nodes to restart bus communications cleanly after an error has occurred.

2.5 Overload Frame

An Overload Frame, shown in Figure 2-5, has the same format as an active error frame. An overload frame, however can only be generated during an Interframe space. In 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 two 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 (and, as for an active error flag, 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 two conditions. First, the node detects a dominant bit during the 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 two sequential overload frames to delay the start of the next message.

2.6 Interframe Space

The Interframe Space separates a preceeding frame (of any type) from a subsequent data or remote frame. The interframe space is composed of at least three recessive bits called the Intermission. This is provided to allow nodes time for internal processing 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.

© 2007 Microchip Technology Inc.

FIGURE 2-1: STANDARD DATA FRAME

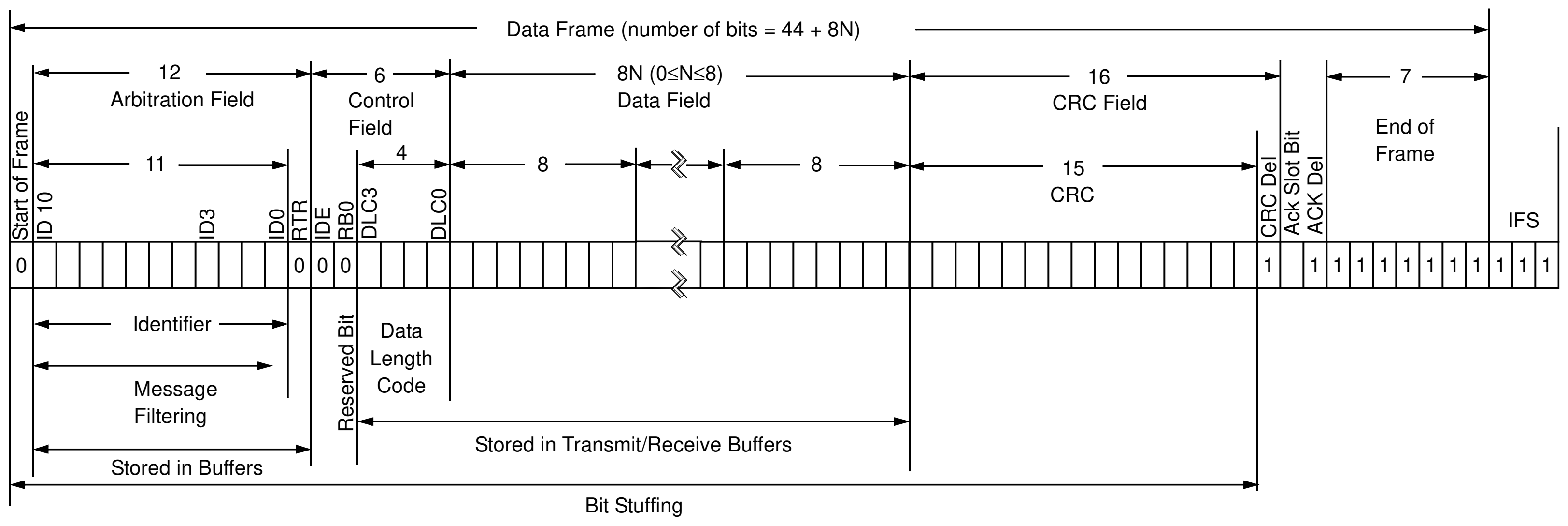


FIGURE 2-2: EXTENDED DATA FRAME

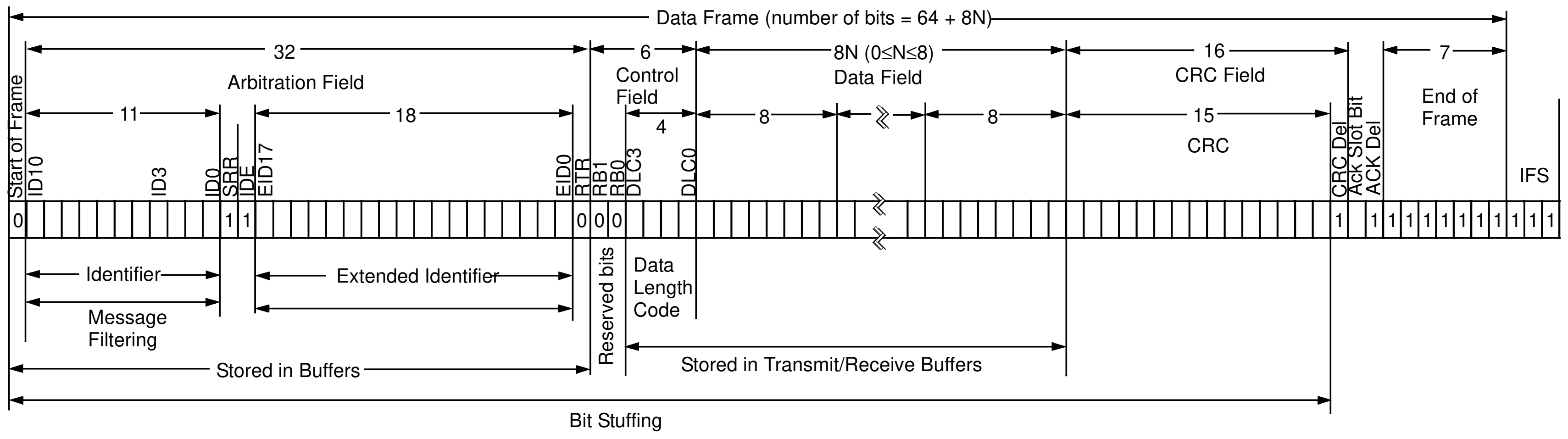
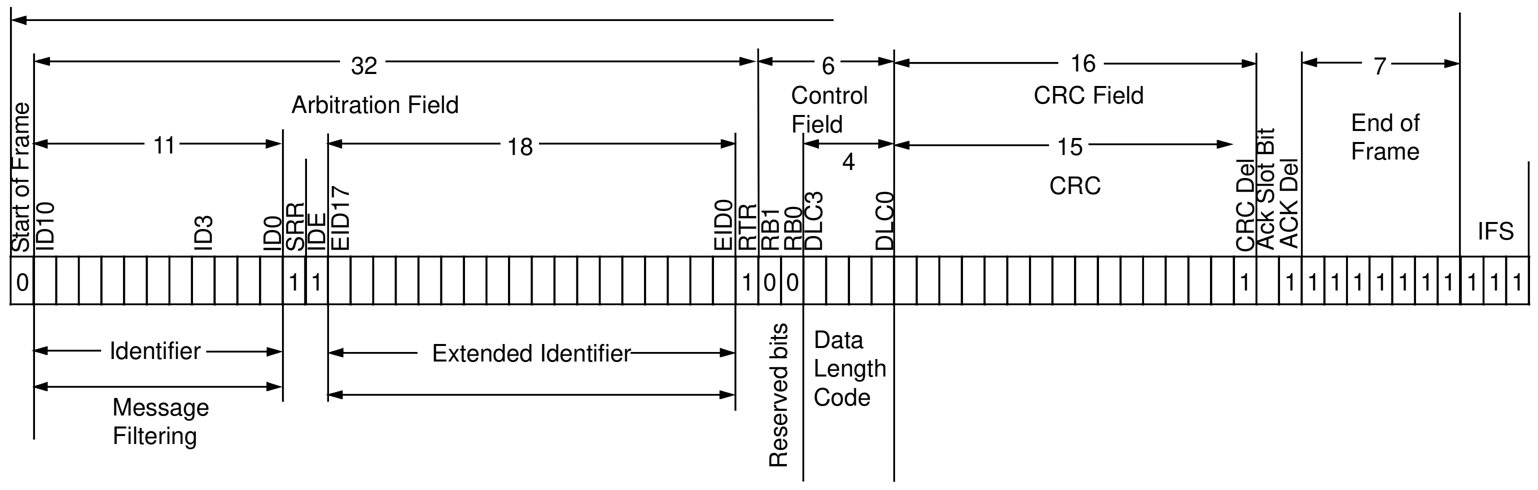


FIGURE 2-3: REMOTE DATA FRAME



Remote Data Frame with Extended Identifier

FIGURE 2-4: ERROR DATA FRAME

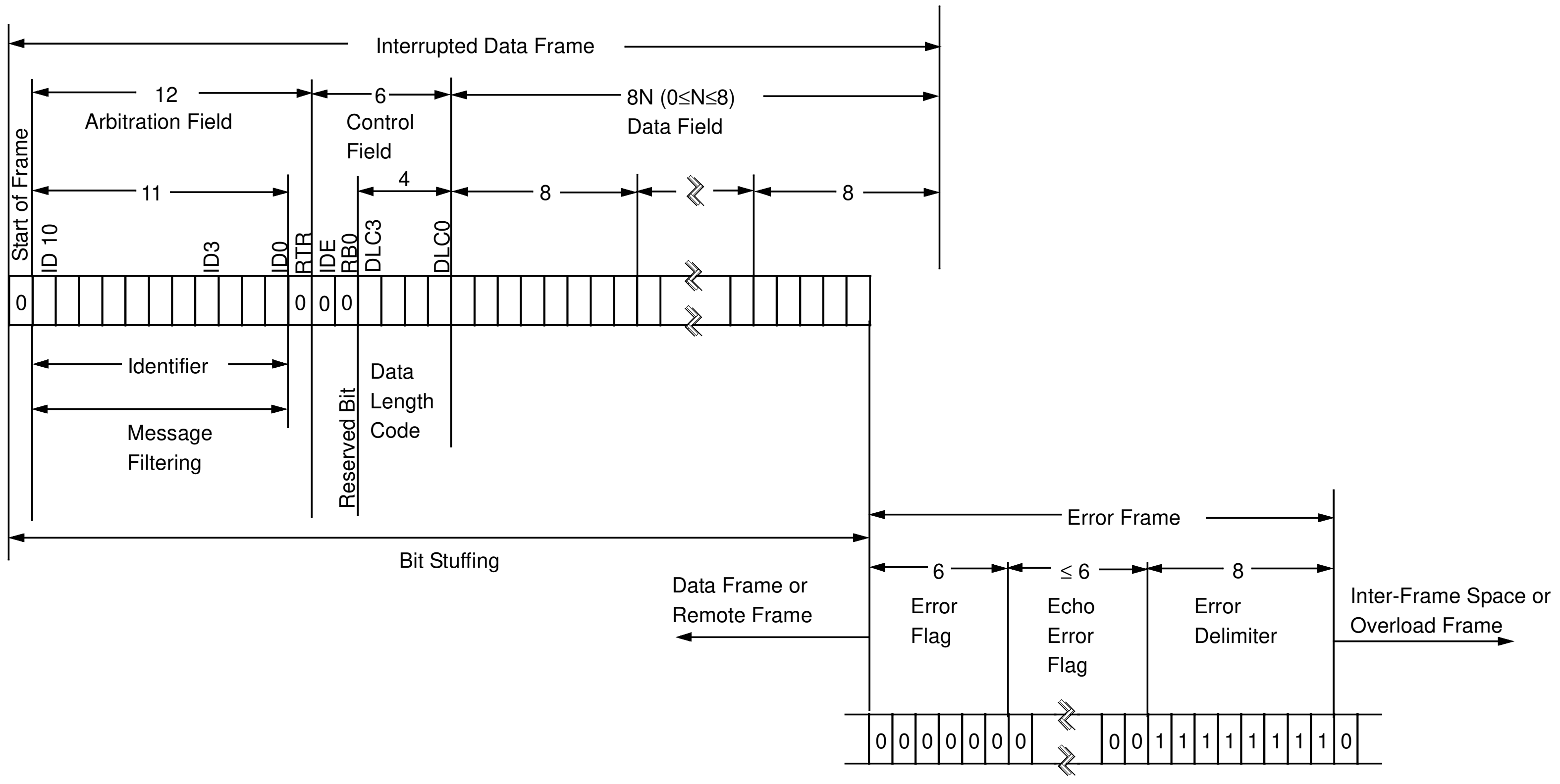
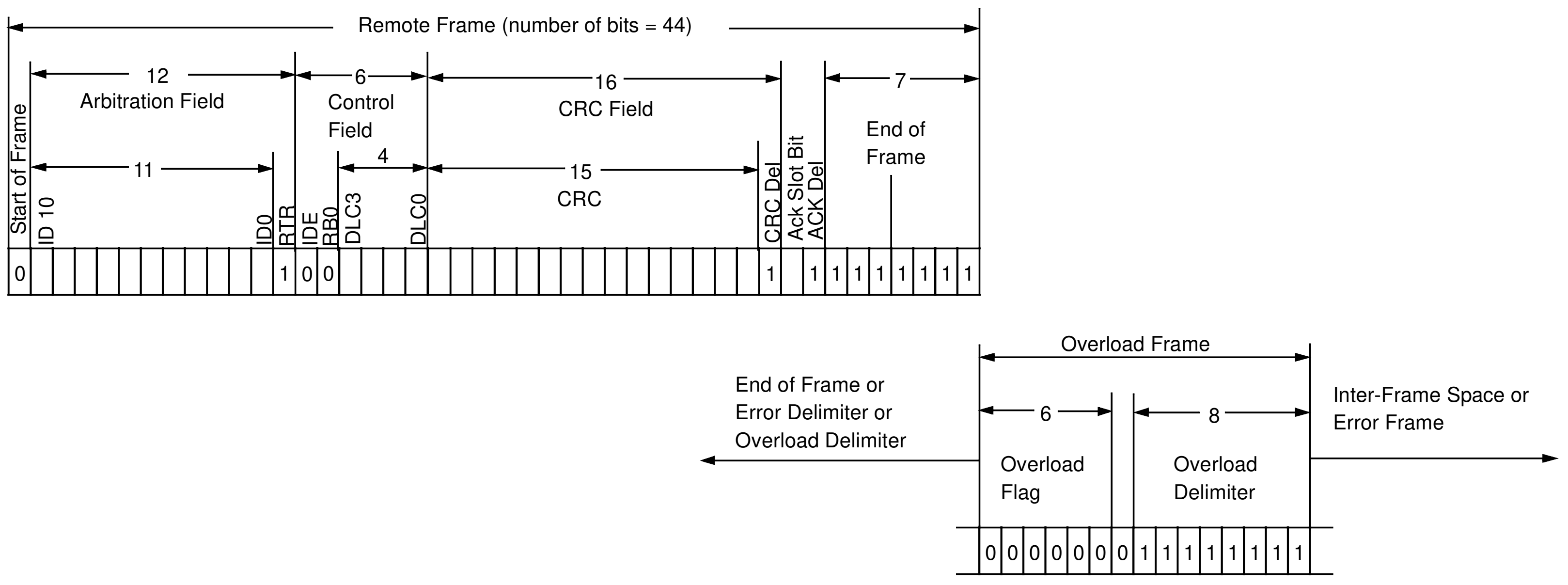


FIGURE 2-5: OVERLOAD FRAME



NOTES:

3.0 MESSAGE TRANSMISSION

3.1 Transmit Buffers

The MCP2510 implements three Transmit Buffers. Each of these buffers occupies 14 bytes of SRAM and are mapped into the device memory maps. The first byte, TXBNCTRL, is a control register associated with the message buffer. The information in this register determines the conditions under which the message will be transmitted and indicates the status of the message transmission. (see Register 3-2). Five bytes are used to hold the standard and extended identifiers and other message arbitration information (see Register 3-3 through Register 3-8). The last eight bytes are for the eight possible data bytes of the message to be transmitted (see Register 3-8).

For the MCU to have write access to the message buffer, the TXBNCTRL.TXREQ bit must be clear, indicating that the message buffer is clear of any pending message to be transmitted. At a minimum, the TXBNSIDH, TXBNSIDL, and TXBNDLC registers must be loaded. If data bytes are present in the message, the TXBNDm registers must also be loaded. If the message is to use extended identifiers, the TXBNEIDm registers must also be loaded and the TXBNSIDL.EXIDE bit set.

Prior to sending the message, the MCU must initialize the CANINTE.TXINE bit to enable or disable the generation of an interrupt when the message is sent. The MCU must also initialize the TXBNCTRL.TXP priority bits (see Section 3.2).

3.2 Transmit Priority

Transmit priority is a prioritization, within the MCP2510, of the pending transmittable messages. This is independent from, and not necessarily related to, any prioritization implicit in the message arbitration scheme built into the CAN protocol. Prior to sending the SOF, the priority of all buffers that are queued for transmission is 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 buffer number will be sent first. For example, if transmit buffer 1 has the same priority setting as transmit buffer 0, buffer 1 will be sent first. There are four levels of transmit priority. If TXBNCTRL.TXP<1:0> for a particular message buffer is set to 11, that buffer has the highest possible priority. If TXBNCTRL.TXP<1:0> for a particular message buffer is 00, that buffer has the lowest possible priority.

3.3 Initiating Transmission

To initiate message transmission the TXBNCTRL.TXREQ bit must be set for each buffer to be transmitted. This can be done by writing to the register via the SPI interface or by setting the $\overline{\text{TXNRTS}}$ pin low for the particular transmit buffer(s) that are to be transmitted. If transmission is initiated via the SPI interface, the TXREQ bit can be set at the same time as the TXP priority bits.

When TXBNCTRL.TXREQ is set, the TXBNCTRL.ABTF, TXBNCTRL.MLOA and TXBNCTRL.TXERR bits will be cleared.

Setting the TXBNCTRL.TXREQ bit does not initiate a message transmission, it merely flags a message buffer as ready for transmission. Transmission will start when the device detects that the bus is available. The device will then begin transmission of the highest priority message that is ready.

When the transmission has completed successfully the TXBNCTRL.TXREQ bit will be cleared, the CANINTF.TXNIF bit will be set, and an interrupt will be generated if the CANINTE.TXNIE bit is set.

If the message transmission fails, the TXBNCTRL.TXREQ will remain set indicating that the message is still pending for transmission and one of the following condition flags will be set. If the message started to transmit but encountered an error condition, the TXBNCTRL. TXERR and the CANINTF.MERRF bits will be set and an interrupt will be generated on the $\overline{\text{INT}}$ pin if the CANINTE.MERRE bit is set. If the message lost arbitration the TXBNCTRL.MLOA bit will be set.

3.4 TXnRTS Pins

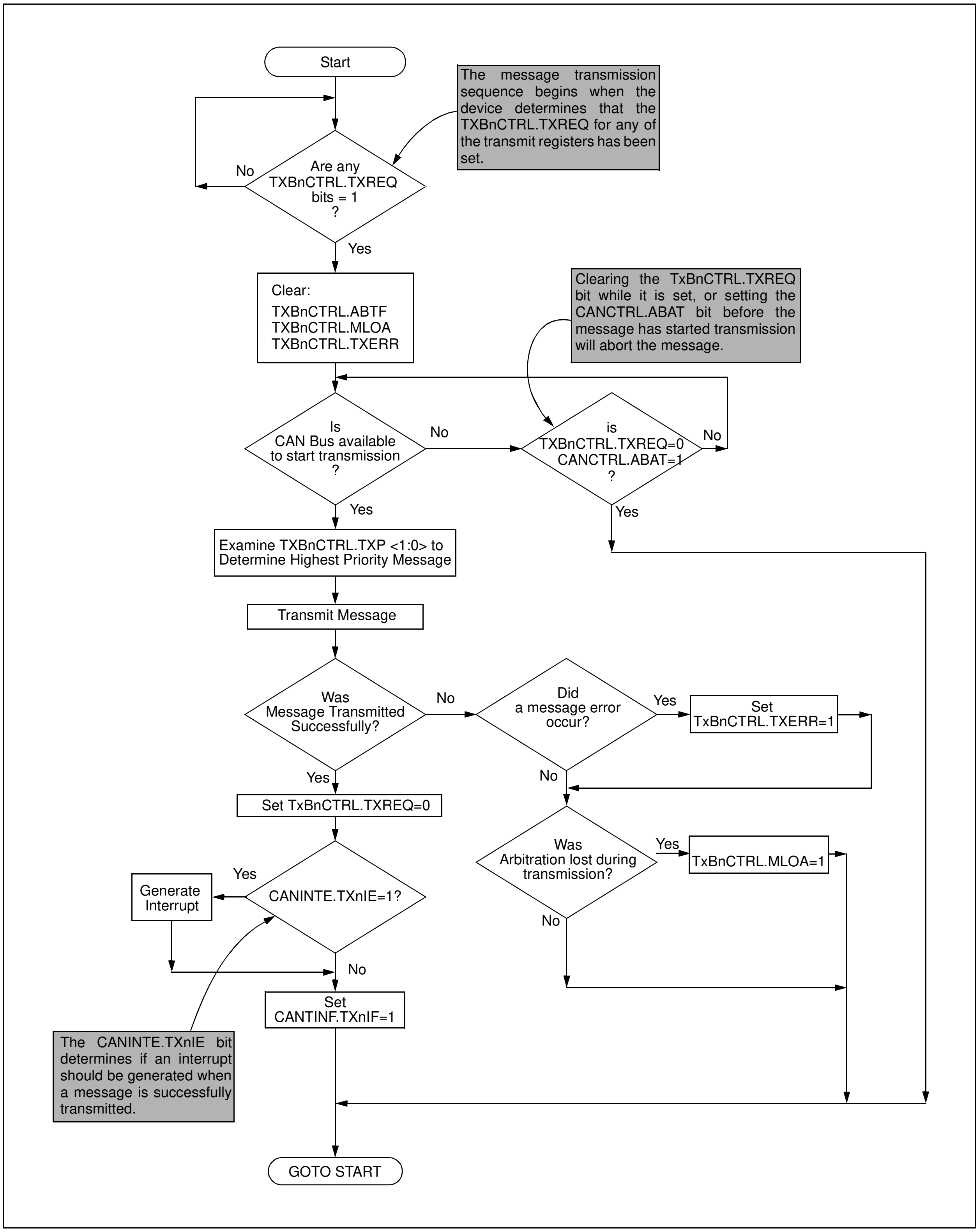
The $\overline{\text{TXNRTS}}$ Pins are input pins that can be configured as request-to-send inputs, which provides a secondary means of initiating the transmission of a message from any of the transmit buffers, or as standard digital inputs. Configuration and control of these pins is accomplished using the TXRTSCTRL register (see Register 3-2). The TXRTSCTRL register can only be modified when the MCP2510 is in configuration mode (see Section 9.0). If configured to operate as a request to send pin, the pin is mapped into the respective TXBNCTRL.TXREQ bit for the transmit buffer. The TXREQ bit is latched by the falling edge of the $\overline{\text{TXNRTS}}$ pin. The $\overline{\text{TXNRTS}}$ pins are designed to allow them to be tied directly to the $\overline{\text{RXNBF}}$ pins to automatically initiate a message transmission when the $\overline{\text{RXNBF}}$ pin goes low. The $\overline{\text{TXNRTS}}$ pins have internal pullup resistors of 100 kΩ (nominal).

3.5 Aborting Transmission

The MCU can request to abort a message in a specific message buffer by clearing the associated TXBnCTRL.TXREQ bit. Also, all pending messages can be requested to be aborted by setting the CANCTRL.ABAT bit. If the CANCTRL.ABAT bit is set to abort all pending messages, the user MUST reset this bit (typically after the user verifies that all TXREQ bits have been cleared) to continue trasmit messages. The CANCTRL.ABTF flag will only be set if the abort was requested via the CANCTRL.ABAT bit. Aborting a message by resetting the TXREQ bit does cause the ATBF bit to be set.

Only messages that have not already begun to be transmitted can be aborted. Once a message has begun transmission, it will not be possible for the user to reset the TXBnCTRL.TXREQ bit. After transmission of a message has begun, if an error occurs on the bus or if the message loses arbitration, the message will be retransmitted regardless of a request to abort.

FIGURE 3-1: TRANSMIT MESSAGE FLOWCHART


Start
The message transmission sequence begins when the device determines that the TXBnCTRL.TXREQ for any of the transmit registers has been set.
No
Are any TXBnCTRL.TXREQ bits = 1 ?
Yes
Clear: TXBnCTRL.ABTF TXBnCTRL.MLOA TXBnCTRL.TXERR
Clearing the TxBnCTRL.TXREQ bit while it is set, or setting the CANCTRL.ABAT bit before the message has started transmission will abort the message.
Is CAN Bus available to start transmission ?
No
is TXBnCTRL.TXREQ=0 CANCTRL.ABAT=1 ?
No
Yes
Yes
Examine TXBnCTRL.TXP <1:0> to Determine Highest Priority Message
Transmit Message
Was Message Transmitted Successfully?
No
Did a message error occur?
Yes
Set TxBnCTRL.TXERR=1
No
Yes
Set TxBnCTRL.TXREQ=0
Was Arbitration lost during transmission?
Yes
TxBnCTRL.MLOA=1
No
Generate Interrupt
Yes
CANINTE.TXnIE=1?
No
Set CANTINF.TXnIF=1
The CANINTE.TXnIE bit determines if an interrupt should be generated when a message is successfully transmitted.
GOTO START

REGISTER 3-1: TXBnCTRL Transmit Buffer n Control Register (ADDRESS: 30h, 40h, 50h)

U-0	R-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
—	ABTF	MLOA	TXERR	TXREQ	—	TXP1	TXP0
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6	**ABTF**: Message Aborted Flag 1 = Message was aborted 0 = Message completed transmission successfully
bit 5	**MLOA**: Message Lost Arbitration 1 = Message lost arbitration while being sent 0 = Message did not lose arbitration while being sent
bit 4	**TXERR**: Transmission Error Detected 1 = A bus error occurred while the message was being transmitted 0 = No bus error occurred while the message was being transmitted
bit 3	**TXREQ**: Message Transmit Request 1 = Buffer is currently pending transmission (MCU sets this bit to request message be transmitted - bit is automatically cleared when the message is sent) 0 = Buffer is not currently pending transmission (MCU can clear this bit to request a message abort)
bit 2	**Unimplemented:** Read as '0'
bit 1-0	**TXP<1:0>**: Transmit Buffer Priority 11 = Highest Message Priority 10 = High Intermediate Message Priority 11 = 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 3-2: TXRTSCTRL - TXnRTS PIN CONTROL AND STATUS REGISTER (ADDRESS: 0Dh)

U-0	U-0	R-x	R-x	R-x	R/W-0	R/W-0	R/W-0
—	—	B2RTS	B1RTS	B0RTS	B2RTSM	B1RTSM	B0RTSM
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6	**Unimplemented:** Read as '0'
bit 5	**B2RTS**: $\overline{\text{TX2RTS}}$ Pin State - Reads state of $\overline{\text{TX2RTS}}$ pin when in digital input mode - Reads as ‘0’ when pin is in ‘request to send’ mode
bit 4	**B1RTS**: $\overline{\text{TX1RTX}}$ Pin State - Reads state of $\overline{\text{TX1RTS}}$ pin when in digital input mode - Reads as ‘0’ when pin is in ‘request to send’ mode
bit 3	**B0RTS**: $\overline{\text{TX0RTS}}$ Pin State - Reads state of $\overline{\text{TX0RTS}}$ pin when in digital input mode - Reads as ‘0’ when pin is in ‘request to send’ mode
bit 2	**B2RTSM**: $\overline{\text{TX2RTS}}$ Pin Mode `1` = Pin is used to request message transmission of TXB2 buffer (on falling edge) `0` = Digital input
bit 1	**B1RTSM**: $\overline{\text{TX1RTS}}$ Pin Mode `1` = Pin is used to request message transmission of TXB1 buffer (on falling edge) `0` = Digital input
bit 0	**B0RTSM**: $\overline{\text{TX0RTS}}$ Pin Mode `1` = Pin is used to request message transmission of TXB0 buffer (on falling edge) `0` = Digital input

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 3-3: TXBnSIDH - TRANSMIT BUFFER n STANDARD IDENTIFIER HIGH (ADDRESS: 31h, 41h, 51h)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0	**SID<10:3>**: Standard Identifier Bits <10:3>

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

© 2007 Microchip Technology Inc.

REGISTER 3-4: TXBnSIDL - Transmit Buffer n Standard Identifier Low (ADDRESS: 32h, 42h, 52h)

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

bit 7-5	**SID<2:0>**: Standard Identifier Bits <2:0>
bit 4	**Unimplemented**: Reads as '0'
bit 3	**EXIDE**: Extended Identifier Enable 1 = Message will transmit extended identifier 0 = Message will transmit standard identifier
bit 2	**Unimplemented**: Reads as '0'
bit 1-0	**EID<17:16>**: Extended Identifier Bits <17:16>

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 3-5: TXBnEID8 - TRANSMIT BUFFER n EXTENDED IDENTIFIER HIGH (ADDRESS: 33h, 43h, 53h)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 7							bit 0

bit 7-0	**EID<15:8>**: Extended Identifier Bits <15:8>

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 3-6: TXBnEID0 - TRANSMIT BUFFER n EXTENDED IDENTIFIER LOW (ADDRESS: 34h, 44h, 54h)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

bit 7-0	**EID<7:0>**: Extended Identifier Bits <7:0>

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

REGISTER 3-7: TXBnDLC - Transmit Buffer n Data Length Code (ADDRESS: 35h, 45h, 55h)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	RTR	—	—	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

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

bit 6 **RTR**: Remote Transmission Request Bit

1 = Transmitted Message will be a Remote Transmit Request
0 = Transmitted Message will be a Data Frame

bit 5-4 **Unimplemented**: Reads as '0'

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

Sets the number of data bytes to be transmitted (0 to 8 bytes)

Note: It is possible to set the DLC to a value greater than 8, however only 8 bytes are transmitted

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 3-8: TXBnDm - Transmit Buffer n Data Field Byte m (ADDRESS: 36h-3Dh, 46h-4Dh, 56h-5Dh)

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
TXBnDm 7	TXBnDm 6	TXBnDm 5	TXBnDm 4	TXBnDm 3	TXBnDm 2	TXBnDm 1	TXBnDm 0
bit 7							bit 0

bit 7-0 **TXBnDm7:TXBnDm0**: Transmit Buffer n Data Field Byte m

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

© 2007 Microchip Technology Inc.

4.0 MESSAGE RECEPTION

4.1 Receive Message Buffering

The MCP2510 includes two full receive buffers with multiple acceptance filters for each. There is also a separate Message Assembly Buffer (MAB) which acts as a third receive buffer (see Figure 4-1).

4.2 Receive Buffers

Of the three Receive Buffers, the MAB is always committed to receiving the next message from the bus. The remaining two receive buffers are called RXB0 and RXB1 and can receive a complete message from the protocol engine. The MCU can access one buffer while the other buffer is available for message reception or holding a previously received message.

The MAB assembles all messages received. These messages will be transferred to the RXBN buffers (See Register 4-4 to Register 4-9) only if the acceptance filter criteria are met.

Note:	The entire contents of the MAB is moved into the receive buffer once a message is accepted. This means that regardless of the type of identifier (standard or extended) and the number of data bytes received, the entire receive buffer is overwritten with the MAB contents. Therefore the contents of all registers in the buffer must be assumed to have been modified when any message is received.

When a message is moved into either of the receive buffers the appropriate CANINTF.RXNIF bit is set. This bit must be cleared by the MCU, when it has completed processing the message in the buffer, in order to allow a new message to be received into the buffer. This bit provides a positive lockout to ensure that the MCU has finished with the message before the MCP2510 attempts to load a new message into the receive buffer. If the CANINTE.RXNIE bit is set an interrupt will be generated on the $\overline{\text{INT}}$ pin to indicate that a valid message has been received.

4.3 Receive Priority

RXB0 is the higher priority buffer and has two message acceptance filters associated with it. RXB1 is the lower priority buffer and has four acceptance filters associated with it. The lower number of acceptance filters makes the match on RXB0 more restrictive and implies a higher priority for that buffer. Additionally, the RXB0CTRL register can be configured such that if RXB0 contains a valid message, and another valid message is received, an overflow error will not occur and the new message will be moved into RXB1 regardless of the acceptance criteria of RXB1. There are also two programmable acceptance filter masks available, one for each receive buffer (see Section 4.5).

When a message is received, bits <3:0> of the RXBNCTRL Register will indicate the acceptance filter number that enabled reception, and whether the received message is a remote transfer request.

The RXBNCTRL.RXM bits set special receive modes. Normally, these bits are set to 00 to enable reception of all valid messages as determined by the appropriate acceptance filters. In this case, the determination of whether or not to receive standard or extended messages is determined by the RFXNSIDL.EXIDE bit in the acceptance filter register. If the RXBNCTRL.RXM bits are set to 01 or 10, the receiver will accept only messages with standard or extended identifiers respectively. If an acceptance filter has the RFXNSIDL.EXIDE bit set such that it does not correspond with the RXBNCTRL.RXM mode, that acceptance filter is rendered useless. These two modes of RXBNCTRL.RXM bits can be used in systems where it is known that only standard or extended messages will be on the bus. If the RXBNCTRL.RXM bits are set to 11, the buffer will receive all messages regardless of the values of the acceptance filters. Also, if a message has an error before the end of frame, that portion of the message assembled in the MAB before the error frame will be loaded into the buffer. This mode has some value in debugging a CAN system and would not be used in an actual system environment.

4.4 RX0BF and RX1BF Pins

In addition to the $\overline{\text{INT}}$ pin which provides an interrupt signal to the MCU for many different conditions, the receive buffer full pins ($\overline{\text{RX0BF}}$ and $\overline{\text{RX1BF}}$) can be used to indicate that a valid message has been loaded into RXB0 or RXB1, respectively.

The $\overline{\text{RXBNBF}}$ full pins can be configured to act as buffer full interrupt pins or as standard digital outputs. Configuration and status of these pins is available via the BFPCTRL register (Register 4-3). When set to operate in interrupt mode (by setting BFPCTRL.BxBFE and BFPCTRL.BxBFM bits to a 1), these pins are active low and are mapped to the CANINTF.RXNIF bit for each receive buffer. When this bit goes high for one of the receive buffers, indicating that a valid message has been loaded into the buffer, the corresponding $\overline{\text{RXNBF}}$ pin will go low. When the CANINTF.RXNIF bit is cleared by the MCU, then the corresponding interrupt pin will go to the logic high state until the next message is loaded into the receive buffer.

When used as digital outputs, the BFPCTRL.BxBFM bits must be cleared to a ‘0’ and BFPCTRL.BxBFE bits must be set to a ‘1’ for the associated buffer. In this mode the state of the pin is controlled by the BFPCTRL.BxBFS bits. Writting a ‘1’ to the BxBFS bit will cause a high level to be driven on the assicated buffer full pin, and a ‘0’ will cause the pin to drive low. When using the pins in this mode the state of the pin should be modified only by using the Bit Modify SPI command to prevent glitches from occuring on either of the buffer full pins.

FIGURE 4-1: RECEIVE BUFFER BLOCK DIAGRAM

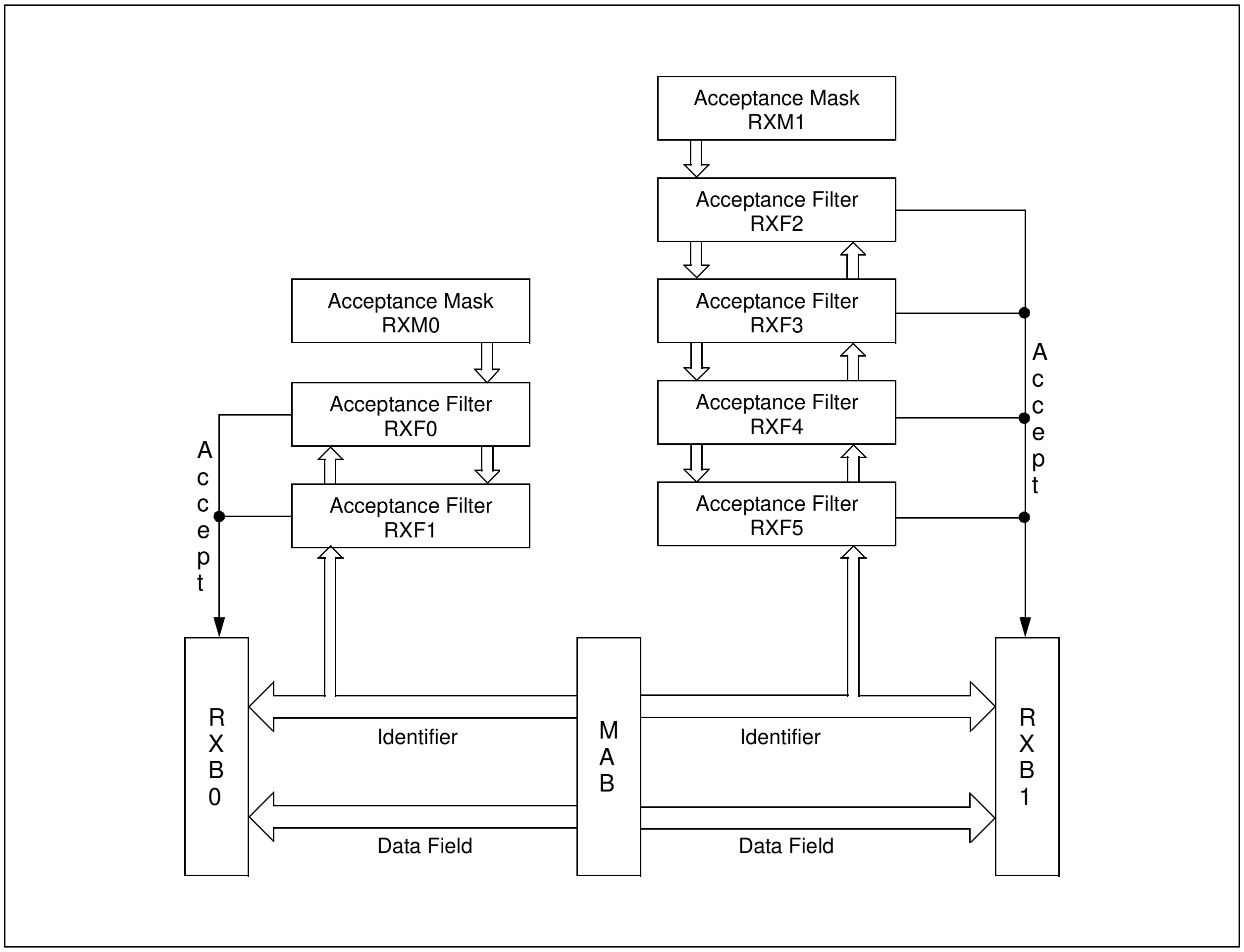
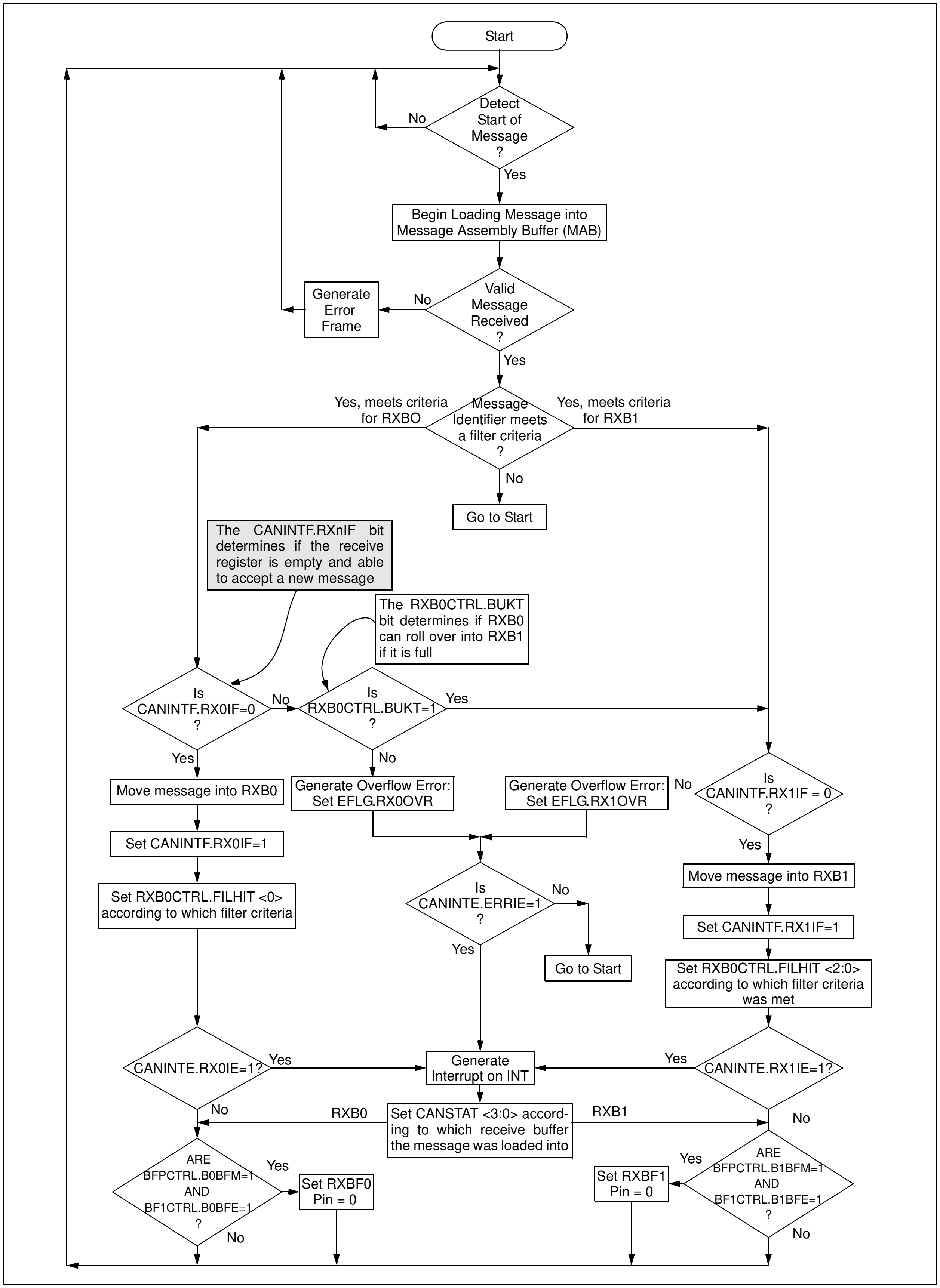


FIGURE 4-2: MESSAGE RECEPTION FLOWCHART


Start
Detect Start of Message ?
No
Yes
Begin Loading Message into Message Assembly Buffer (MAB)
Valid Message Received ?
No
Generate Error Frame
Yes
Message Identifier meets a filter criteria ?
Yes, meets criteria for RXBO
Yes, meets criteria for RXB1
No
Go to Start
The CANINTF.RXnIF bit determines if the receive register is empty and able to accept a new message
The RXB0CTRL.BUKT bit determines if RXB0 can roll over into RXB1 if it is full
Is CANINTF.RX0IF=0 ?
No
Is RXB0CTRL.BUKT=1 ?
Yes
Yes
No
Move message into RXB0
Generate Overflow Error: Set EFLG.RX0OVR
Generate Overflow Error: Set EFLG.RX1OVR
No
Is CANINTF.RX1IF = 0 ?
Yes
Set CANINTF.RX0IF=1
Move message into RXB1
Is CANINTE.ERRIE=1 ?
No
Yes
Set RXB0CTRL.FILHIT <0> according to which filter criteria
Set CANINTF.RX1IF=1
Go to Start
Set RXB0CTRL.FILHIT <2:0> according to which filter criteria was met
CANINTE.RX0IE=1?
Yes
Generate Interrupt on INT
Yes
CANINTE.RX1IE=1?
No
RXB0
Set CANSTAT <3:0> according to which receive buffer the message was loaded into
RXB1
No
ARE BFPCTRL.B0BFM=1 AND BF1CTRL.B0BFE=1 ?
Yes
Set RXBF0 Pin = 0
No
Set RXBF1 Pin = 0
Yes
ARE BFPCTRL.B1BFM=1 AND BF1CTRL.B1BFE=1 ?
No

REGISTER 4-1: RXB0CTRL - RECEIVE BUFFER 0 CONTROL REGISTER (ADDRESS: 60h)

U-0	R/W-0	R/W-0	U-0	R-0	R/W-0	R-0	R-0
—	RXM1	RXM0	—	RXRTR	BUKT	BUKT1	FILHIT0
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6-5	**RXM<1:0>**: Receive Buffer Operating Mode `11` = Turn mask/filters off; receive any message `10` = Receive only valid messages with extended identifiers that meet filter criteria `01` = Receive only valid messages with standard identifiers that meet filter criteria `00` = Receive all valid messages using either standard or extended identifiers that meet filter criteria
bit 4	**Unimplemented:** Read as '0'
bit 3	**RXRTR**: Received Remote Transfer Request `1` = Remote Transfer Request Received `0` = No Remote Transfer Request Received
bit 2	**BUKT**: Rollover Enable `1` = RXB0 message will rollover and be written to RXB1 if RXB0 is full `0` = Rollover disabled
bit 1	**BUKT1**: Read Only Copy of BUKT Bit (used internally by the MCP2510).
bit 0	**FILHIT<0>**: Filter Hit - indicates which acceptance filter enabled reception of message `1` = Acceptance Filter 1 (RXF1) `0` = Acceptance Filter 0 (RXF0) **Note:** If a rollover from RXB0 to RXB1 occurs, the FILHIT bit will reflect the filter that accepted the message that rolled over

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

© 2007 Microchip Technology Inc.

REGISTER 4-2: RXB1CTRL - RECEIVE BUFFER 1 CONTROL REGISTER (ADDRESS: 70h)

U-0	R/W-0	R/W-0	U-0	R-0	R-0	R-0	R-0
—	RXM1	RXM0	—	RXRTR	FILHIT2	FILHIT1	FILHIT0
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6-5	**RXM<1:0>**: Receive Buffer Operating Mode `11` = Turn mask/filters off; receive any message `10` = Receive only valid messages with extended identifiers that meet filter criteria `01` = Receive only valid messages with standard identifiers that meet filter criteria `00` = Receive all valid messages using either standard or extended identifiers that meet filter criteria
bit 4	**Unimplemented:** Read as '0'
bit 3	**RXRTR**: Received Remote Transfer Request `1` = Remote Transfer Request Received `0` = No Remote Transfer Request Received
bit 2-0	**FILHIT<2:0>**: Filter Hit - indicates which acceptance filter enabled reception of message `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 if BUKT bit set in RXB0CTRL) `000` = Acceptance Filter 0 (RXF0) (Only if BUKT bit set in RXB0CTRL)

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