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

MCP2515

Stand-Alone CAN Controller with SPI Interface

Features

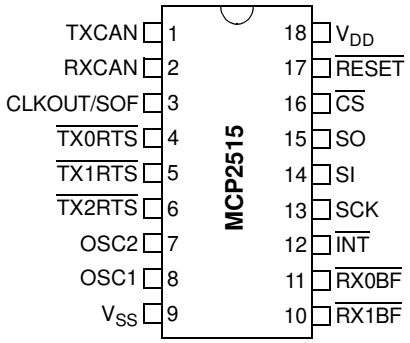
- Implements CAN V2.0B at 1 Mb/s:
  - 0 to 8-byte length in the data field
  - Standard and extended data and remote frames
- Receive Buffers, Masks and Filters:
  - Two receive buffers with prioritized message storage
  - Six 29-bit filters
  - Two 29-bit masks
- Data Byte Filtering on the First Two Data Bytes (applies to standard data frames)
- Three Transmit Buffers with Prioritization and Abort Features
- High-Speed SPI Interface (10 MHz):
  - SPI modes 0,0 and 1,1
- One-Shot mode Ensures Message Transmission is Attempted Only One Time
- Clock Out Pin with Programmable Prescaler:
  - Can be used as a clock source for other device(s)
- Start-of-Frame (SOF) Signal is Available for Monitoring the SOF Signal:
  - Can be used for time slot-based protocols and/or bus diagnostics to detect early bus degradation
- Interrupt Output Pin with Selectable Enables
- Buffer Full Output Pins Configurable as:
  - Interrupt output for each receive buffer
  - General purpose output
- Request-to-Send (RTS) Input Pins Individually Configurable as:
  - Control pins to request transmission for each transmit buffer
  - General purpose inputs
- Low-Power CMOS Technology:
  - Operates from 2.7V-5.5V
  - 5 mA active current (typical)
  - 1 µA standby current (typical) (Sleep mode)
- Temperature Ranges Supported:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

Description

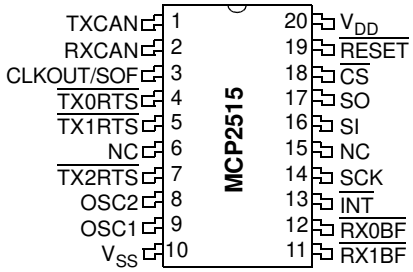
Microchip Technology's MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification, Version 2.0B. It is capable of transmitting and receiving both standard and extended data and remote frames. The MCP2515 has two acceptance masks and six acceptance filters that are used to filter out unwanted messages, thereby reducing the host MCU's overhead. The MCP2515 interfaces with microcontrollers (MCUs) via an industry standard Serial Peripheral Interface (SPI).

Package Types

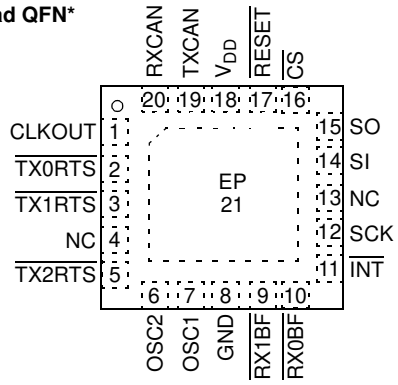
18-Lead PDIP/SOIC



20-Lead TSSOP



20-Lead QFN*



* Includes Exposed Thermal Pad (EP); see Table 1-1.

NOTES:

1.0 DEVICE OVERVIEW

The MCP2515 is a stand-alone CAN controller developed to simplify applications that require interfacing with a CAN bus. A simple block diagram of the MCP2515 is shown in Figure 1-1. The device consists of three main blocks:

1. The CAN module, which includes the CAN protocol engine, masks, filters, transmit and receive buffers.
2. The control logic and registers that are used to configure the device and its operation.
3. The SPI protocol block.

An example system implementation using the device is shown in Figure 1-2.

1.1 CAN Module

The CAN module handles all functions for receiving and transmitting messages on the CAN 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.

1.2 Control Logic

The control logic block controls the setup and operation of the MCP2515 by interfacing to the other blocks in order to pass information and control.

Interrupt pins are provided to allow greater system flexibility. There is one multipurpose interrupt pin (as well as specific interrupt pins) for each of the receive registers that can be used to indicate a valid message has been received and loaded into one of the receive buffers. Use of the specific interrupt pins is optional. 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.

Additionally, there are 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, as initiating message transmissions can also be accomplished by utilizing control registers accessed via the SPI interface.

1.3 SPI Protocol Block

The MCU interfaces to the device via the SPI interface. Writing to, and reading from, all registers is accomplished using standard SPI read and write commands, in addition to specialized SPI commands.

FIGURE 1-1: BLOCK DIAGRAM

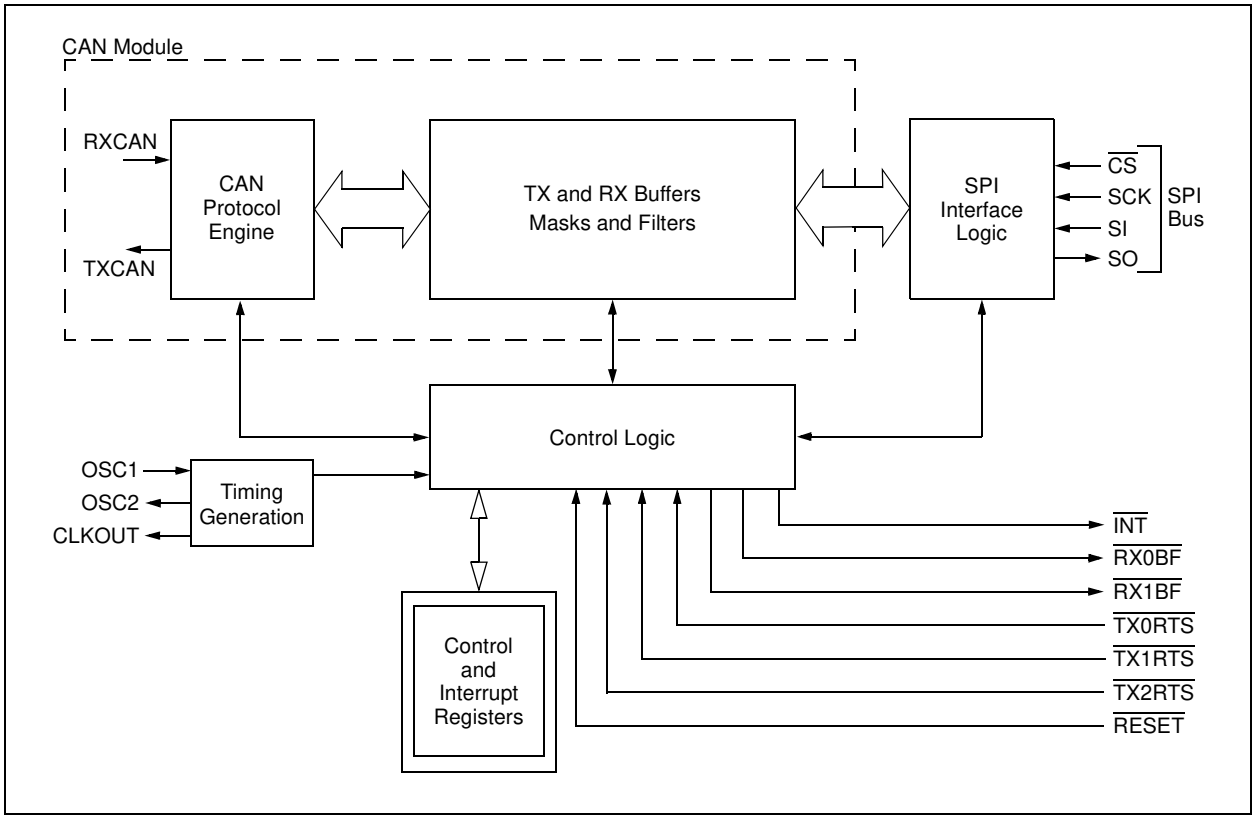


FIGURE 1-2: EXAMPLE SYSTEM IMPLEMENTATION

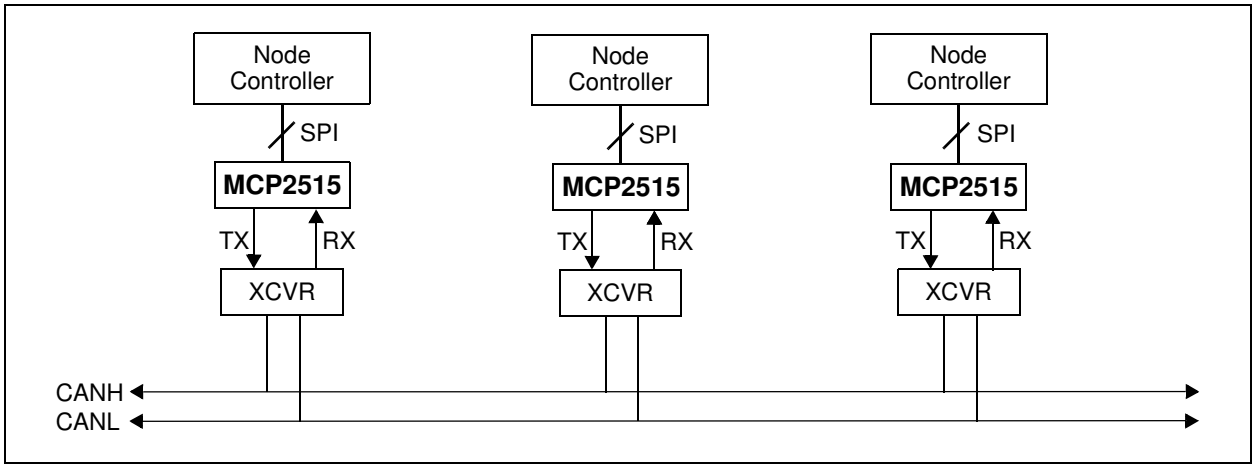


TABLE 1-1: PINOUT DESCRIPTION

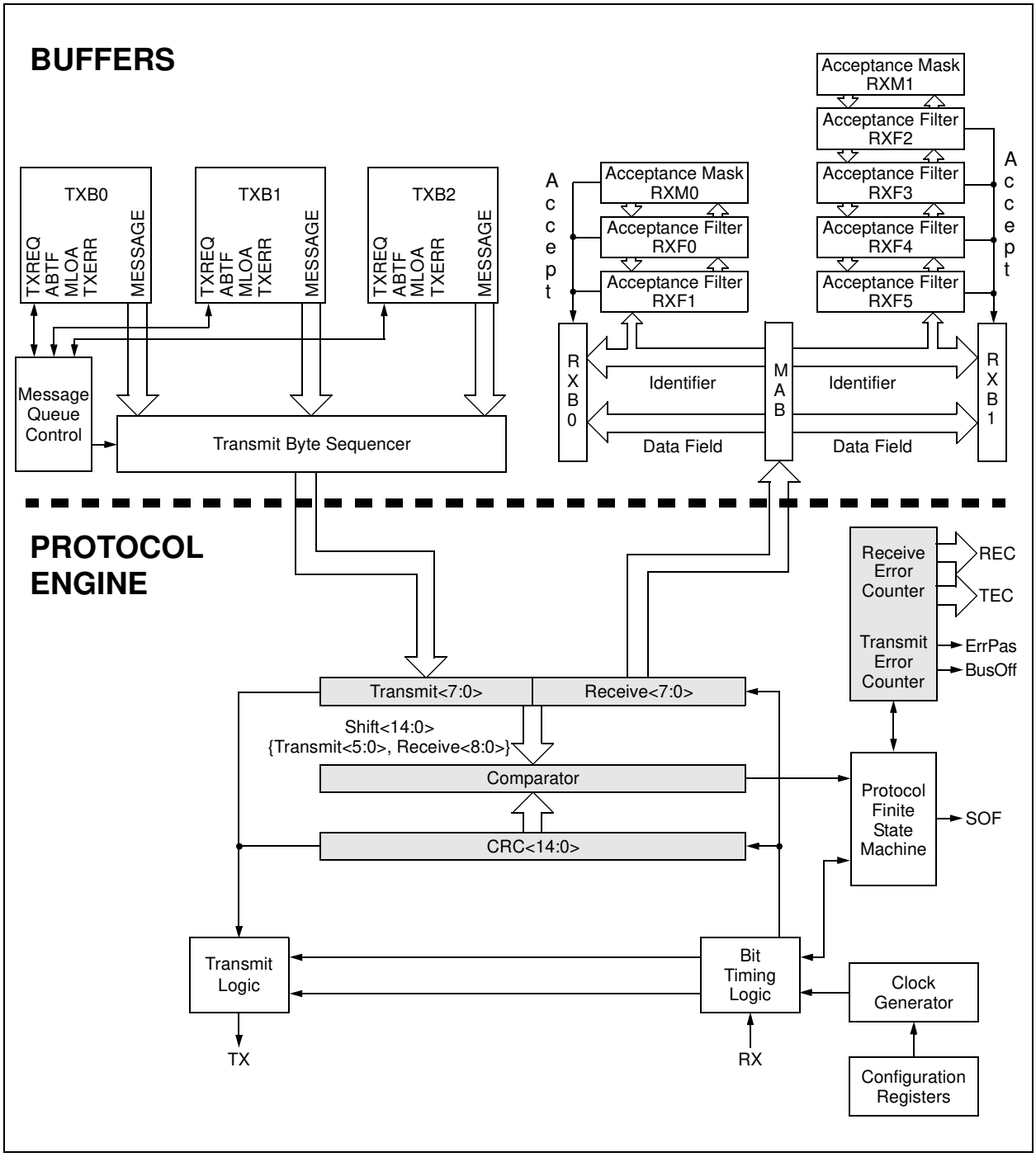
Name	PDIP/ SOIC Pin #	TSSOP Pin #	QFN Pin #	I/O/P Type	Description	Alternate Pin Function
TXCAN	1	1	19	O	Transmit output pin to CAN bus	—
RXCAN	2	2	20	I	Receive input pin from CAN bus	—
CLKOUT	3	3	1	O	Clock output pin with programmable prescaler	Start-of-Frame signal
$\overline{TX0RTS}$	4	4	2	I	Transmit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

Legend: I = Input; O = Output; P = Power

1.4 Transmit/Receive Buffers/Masks/ Filters

The MCP2515 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 shows a block diagram of these buffers and their connection to the protocol engine.

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



1.5 CAN Protocol Engine

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

1.5.1 PROTOCOL FINITE STATE MACHINE

The heart of the engine is the Finite State Machine (FSM). The FSM is a sequencer that controls 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 ensures 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.

1.5.2 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.

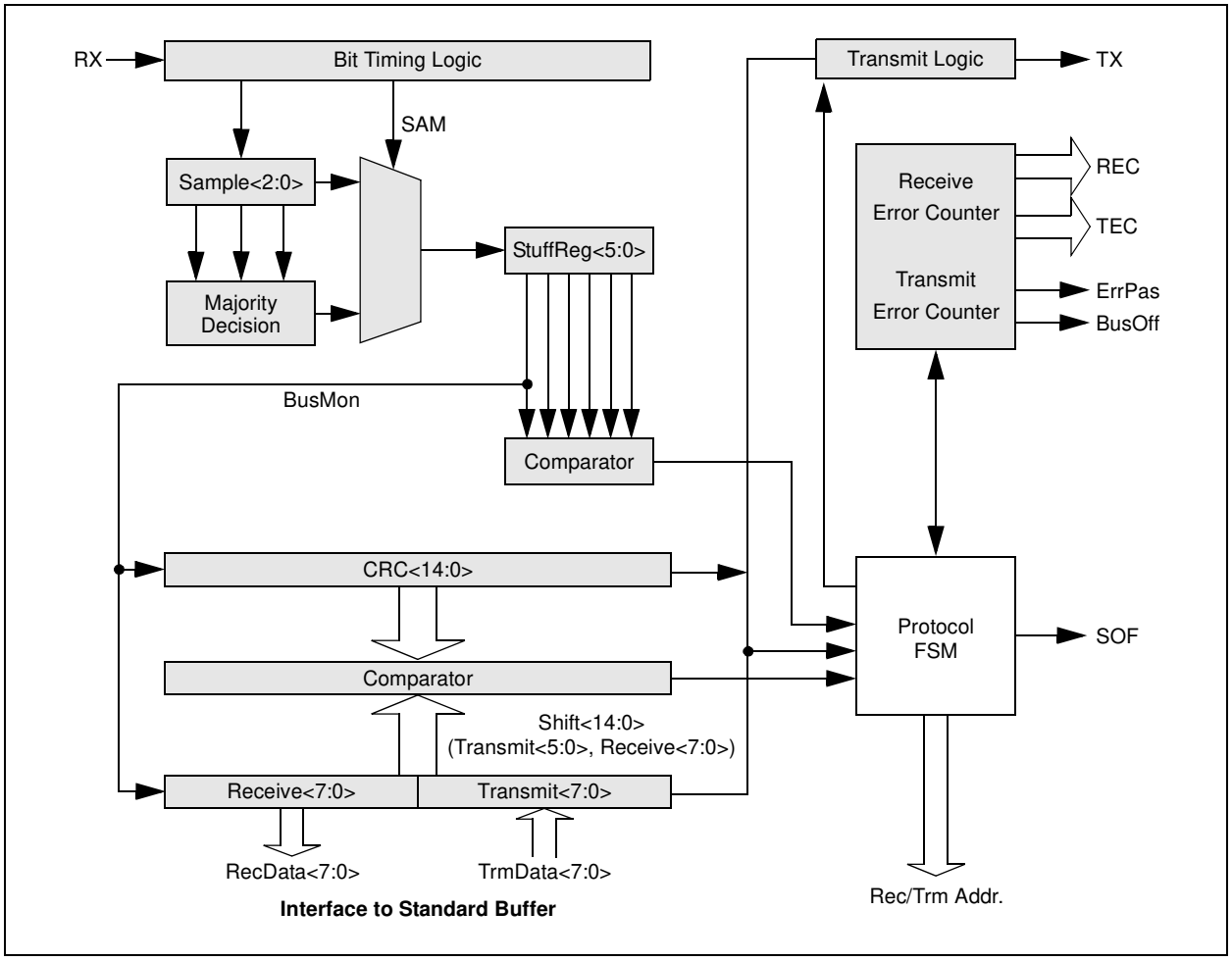
1.5.3 ERROR MANAGEMENT LOGIC

The Error Management Logic (EML) 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. Based on the values of the error counters, the CAN controller is set into the states: error-active, error-passive or bus-off.

1.5.4 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 the 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 on the baud rate and external physical delay times.

FIGURE 1-4: CAN PROTOCOL ENGINE BLOCK DIAGRAM



2.0 CAN MESSAGE FRAMES

The MCP2515 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. As with all other frames, the frame begins with a Start-of-Frame (SOF) bit, which is of the dominant state and 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 as 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 (0-8 bytes) 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 (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 (ACK) 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, shown in Figure 2-2, the SOF bit is followed by the arbitration field, which consists of 32 bits. The first 11 bits are the Most Significant bits (MSb) (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), 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, the 29-bit extended message identifier is split 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 the 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 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 a standard data frame (see **Section 2.1 "Standard Data Frame"**).

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. The type of error flag field sent depends upon the error status of the node that detects and generates the error flag field.

2.4.1 ACTIVE ERRORS

If an error-active node detects a bus error, 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 (eight recessive bits) completes the error frame. Upon completion of the error frame, bus activity returns to normal and the interrupted node attempts to resend the aborted message.

Note:	Error echo flags typically occur when a localized disturbance causes one or more (but not all) nodes to send an error flag. The remaining nodes generate error flags in response (echo) to the original error flag.

2.4.2 PASSIVE ERRORS

If an error-passive node detects a bus error, the node transmits an error-passive flag followed by the error delimiter field. The error-passive flag consists of six consecutive recessive bits. 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 an error-active node or the transmitting node, the message will continue transmission because the error-passive flag does not interfere with the bus.

If the transmitting node generates an error-passive flag, it 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:

1. The node detects a dominant bit during the interframe space, an illegal condition. **Exception:** The dominant bit is detected during the third bit of IFS. In this case, the receivers will interpret this as a SOF.
2. Due to internal conditions, the node is not yet able to begin reception of the next message. A node may generate a maximum of two sequential overload frames to delay the start of the next message.

Note:	Case 2 should never occur with the MCP2515 due to very short internal delays.

2.6 Interframe Space

The interframe space separates a preceding 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 allows 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.

FIGURE 2-1: STANDARD DATA FRAME

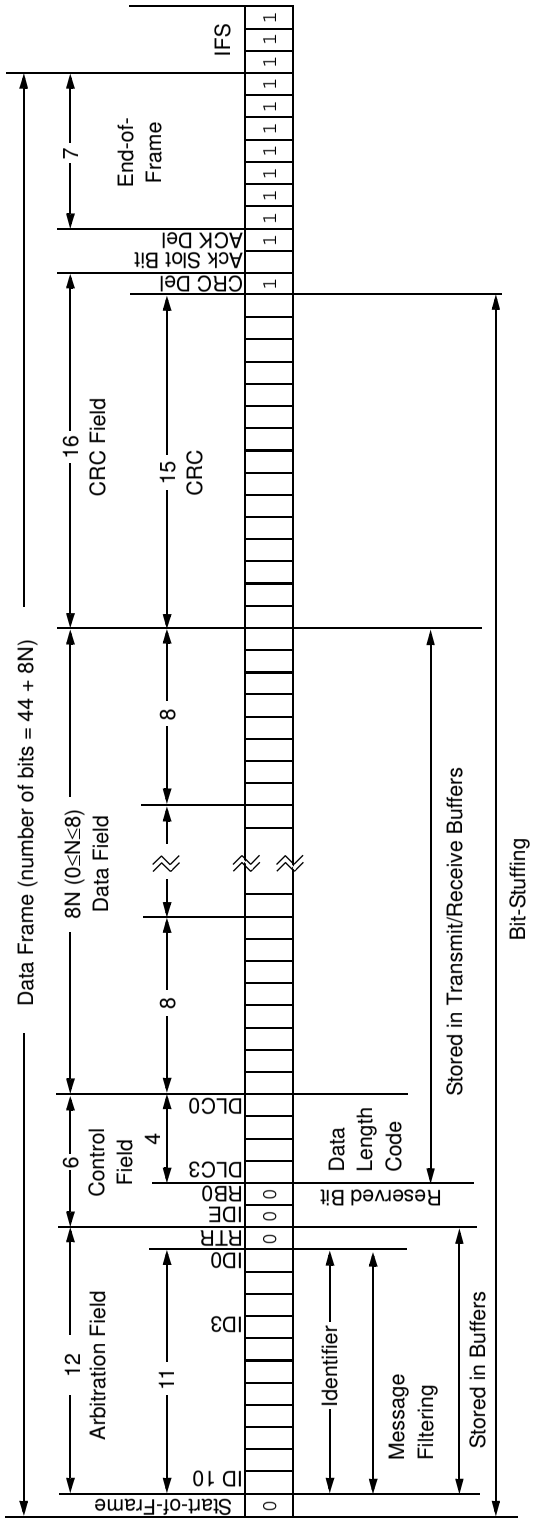


FIGURE 2-2: EXTENDED DATA FRAME

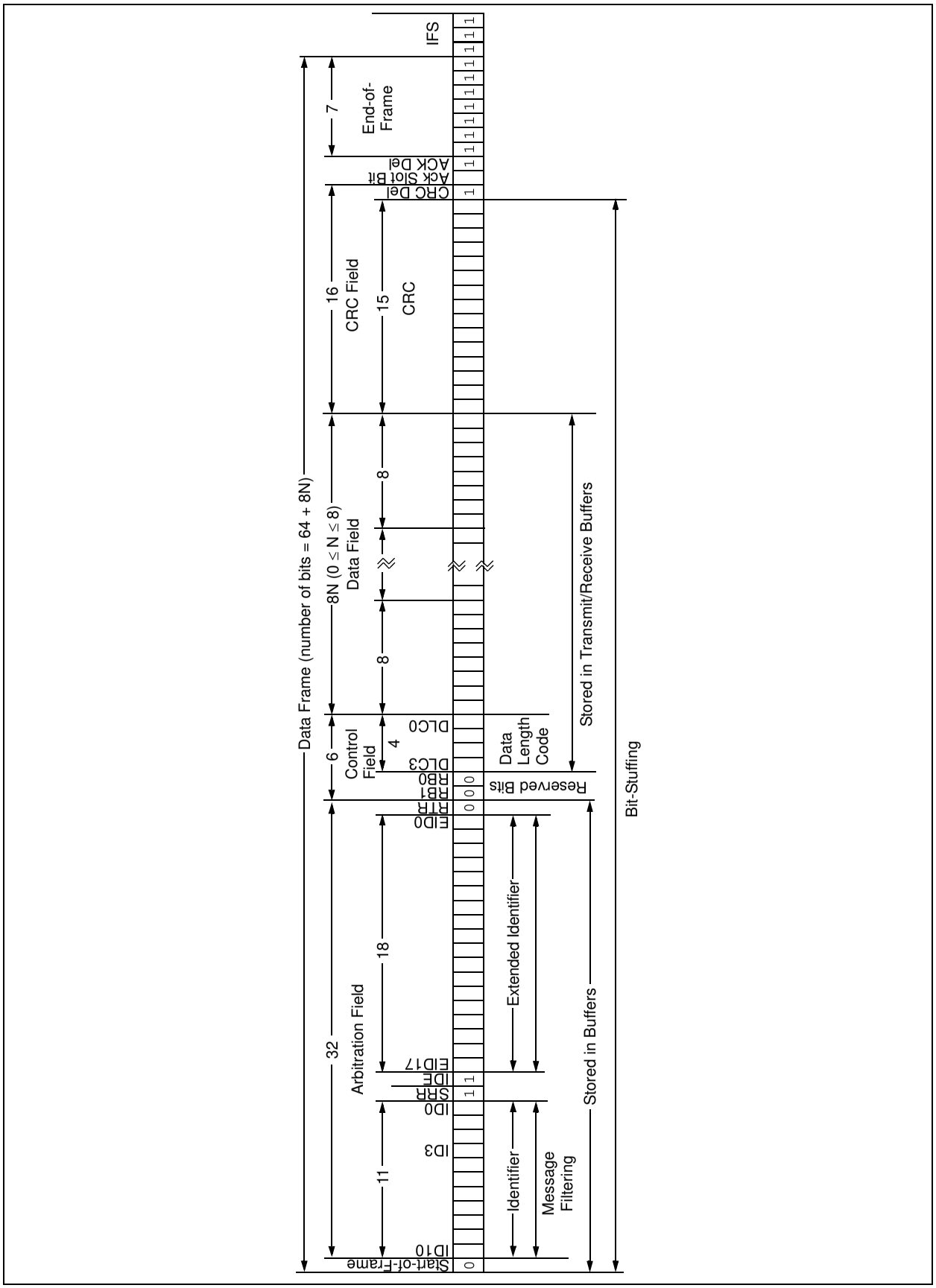


FIGURE 2-3: REMOTE FRAME

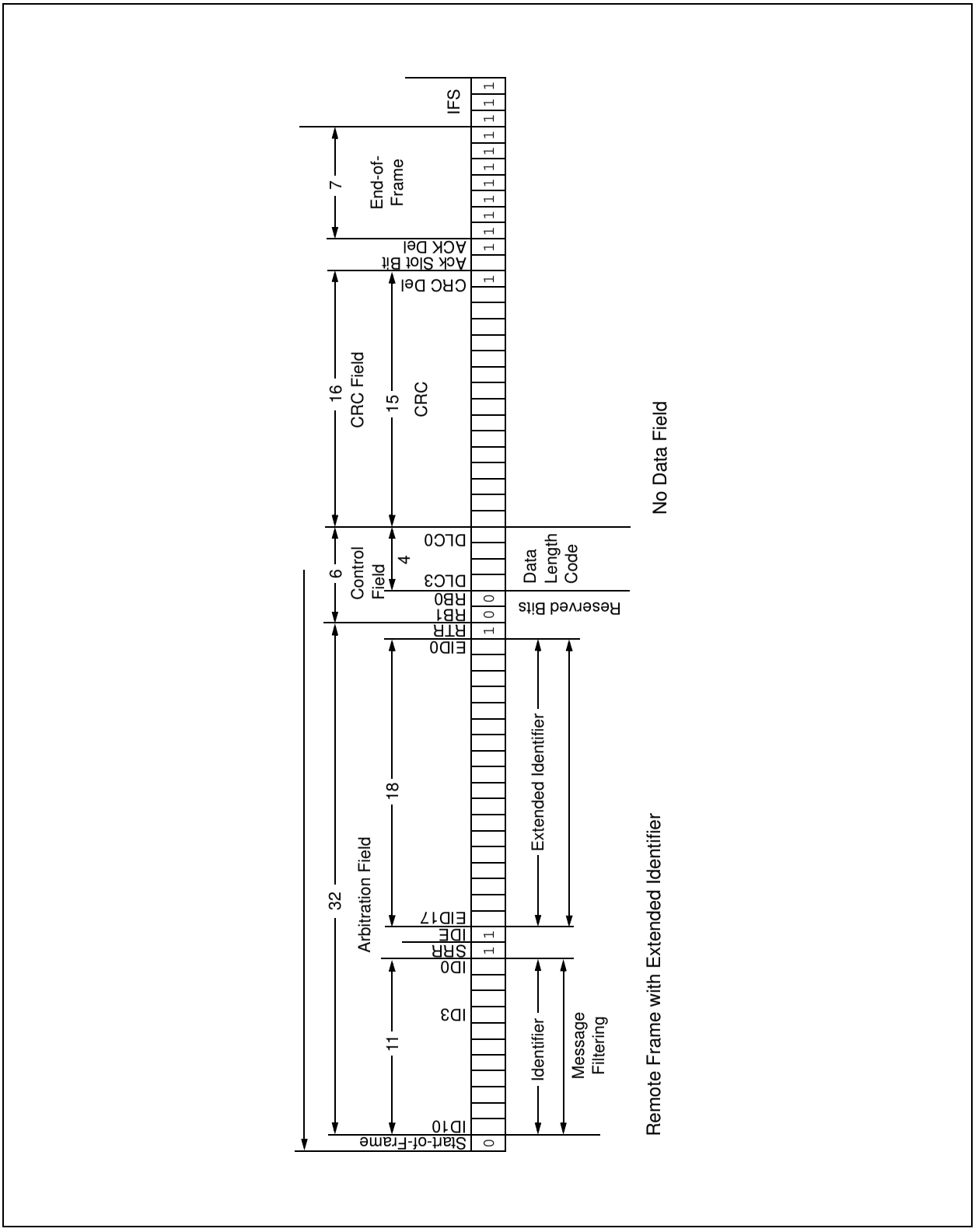


FIGURE 2-4: ACTIVE ERROR FRAME

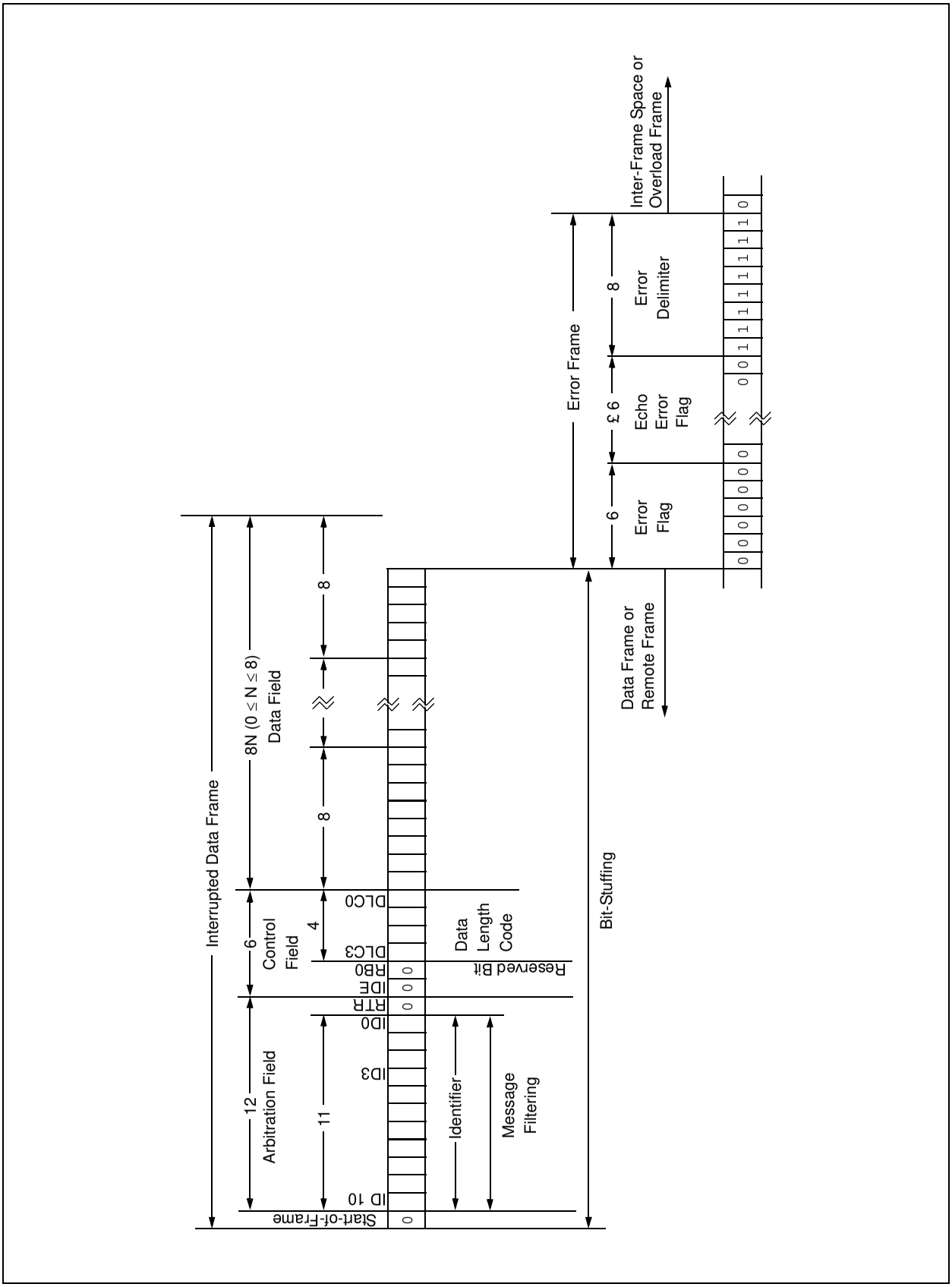
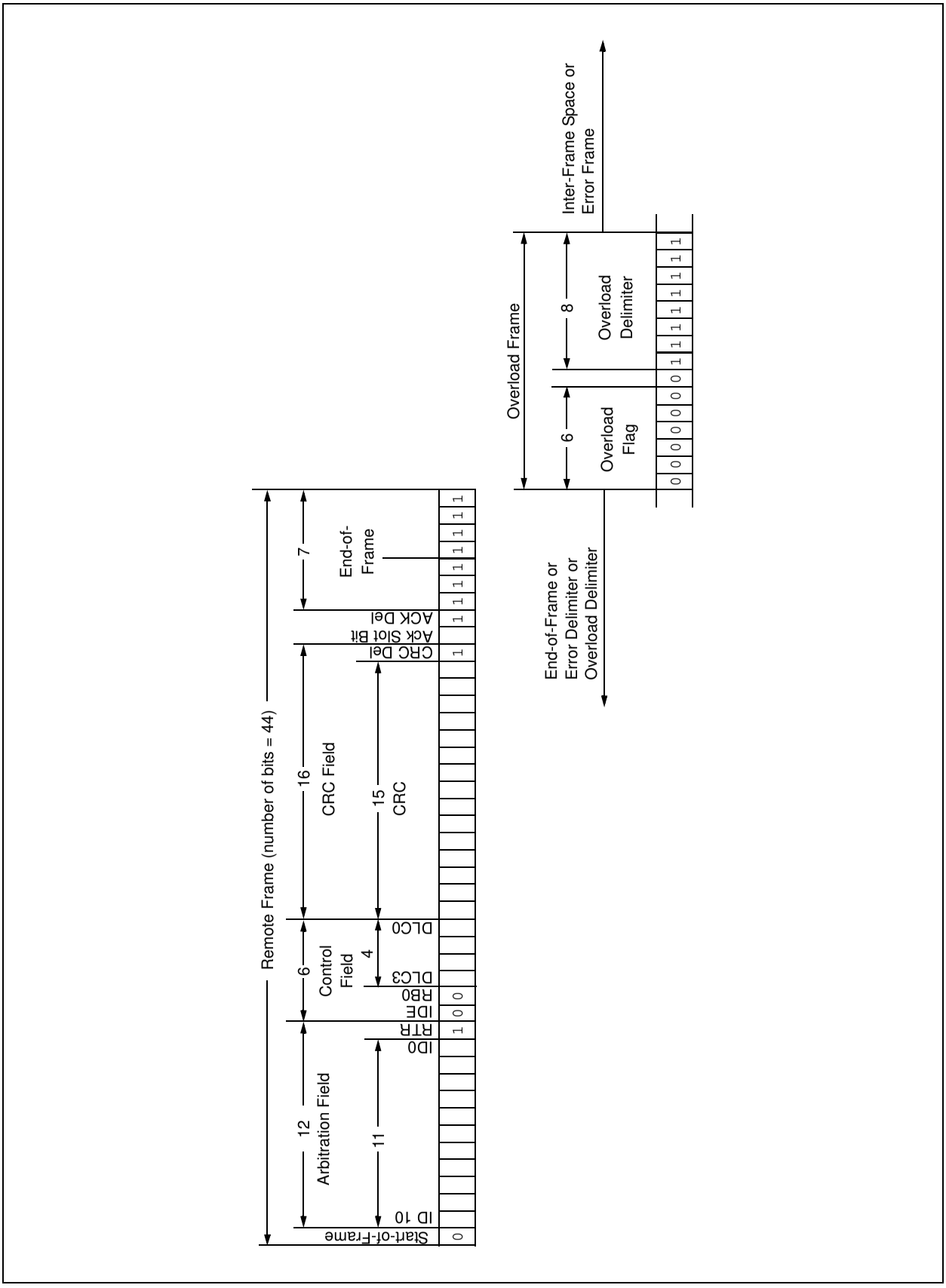


FIGURE 2-5: OVERLOAD FRAME



NOTES:

3.0 MESSAGE TRANSMISSION

3.1 Transmit Buffers

The MCP2515 implements three transmit buffers. Each of these buffers occupies 14 bytes of SRAM and are mapped into the device memory map.

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 3-1).

Five bytes are used to hold the Standard and Extended Identifiers, as well as other message arbitration information (see Register 3-3 through Register 3-6). The last eight bytes are for the eight possible data bytes of the message to be transmitted (see Register 3-8).

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 EXIDE (TXBnSIDL<3>) bit set.

Prior to sending the message, the MCU must initialize the TXnIE bit in the CANINTE register to enable or disable the generation of an interrupt when the message is sent.

Note:	The TXREQ bit (TXBnCTRL<3>) must be clear (indicating the transmit buffer is not pending transmission) before writing to the transmit buffer.

3.2 Transmit Priority

Transmit priority is a prioritization within the MCP2515 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, Transmit 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, Transmit Buffer 1 will be sent first.

There are four levels of transmit priority. If the TXP<1:0> bits (TXBnCTRL<1:0>) for a particular message buffer are set to '11', that buffer has the highest possible priority. If the TXP<1:0> bits for a particular message buffer are '00', that buffer has the lowest possible priority.

3.3 Initiating Transmission

In order to initiate message transmission, the TXREQ bit (TXBnCTRL<3>) must be set for each buffer to be transmitted. This can be accomplished by:

- Writing to the register via the SPI write command
- Sending the SPI RTS command
- 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 TXPx priority bits.

When TXREQ is set, the ABTF, MLOA and TXERR bits (TXBnCTRL<5:4>) will be cleared automatically.

Note:	Setting the TXREQ bit (TXBnCTRL<3>) does not initiate a message transmission. It merely flags a message buffer as being ready for transmission. Transmission will start when the device detects that the bus is available.

Once the transmission has completed successfully, the TXREQ bit will be cleared, the TXnIF bit (CANINTF) will be set and an interrupt will be generated if the TXnIE bit (CANINTE) is set.

If the message transmission fails, the TXREQ bit will remain set. This indicates 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 TXERR (TXBnCTRL<4>) and MERRF bits (CANINTF<7>) will be set, and an interrupt will be generated on the $\overline{\text{INT}}$ pin if the MERRE bit (CANINTE<7>) is set
- If the message is lost, arbitration at the MLOA bit (TXBnCTRL<5>) will be set

Note:	If One-Shot mode is enabled (OSM bit (CANCTRL<3>)), the above conditions will still exist. However, the TXREQ bit will be cleared and the message will not attempt transmission a second time.

3.4 One-Shot Mode

One-Shot mode ensures that a message will only attempt to transmit one time. Normally, if a CAN message loses arbitration, or is destroyed by an error frame, the message is retransmitted. With One-Shot mode enabled, a message will only attempt to transmit one time, regardless of arbitration loss or error frame.

One-Shot mode is required to maintain time slots in deterministic systems, such as TTCAN.

3.5 $\overline{\text{TXnRTS}}$ Pins

The $\overline{\text{TXnRTS}}$ pins are input pins that can be configured as:

- Request-to-Send inputs, which provide an alternative means of initiating the transmission of a message from any of the transmit buffers
- 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 MCP2515 is in Configuration mode (see **Section 10.0 "Modes of Operation"**). If configured to operate as a Request-to-Send pin, the pin is mapped into the respective TXREQ bit (TXBnCTRL<3>) 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 pull-up resistors of 100 kΩ (nominal).

3.6 Aborting Transmission

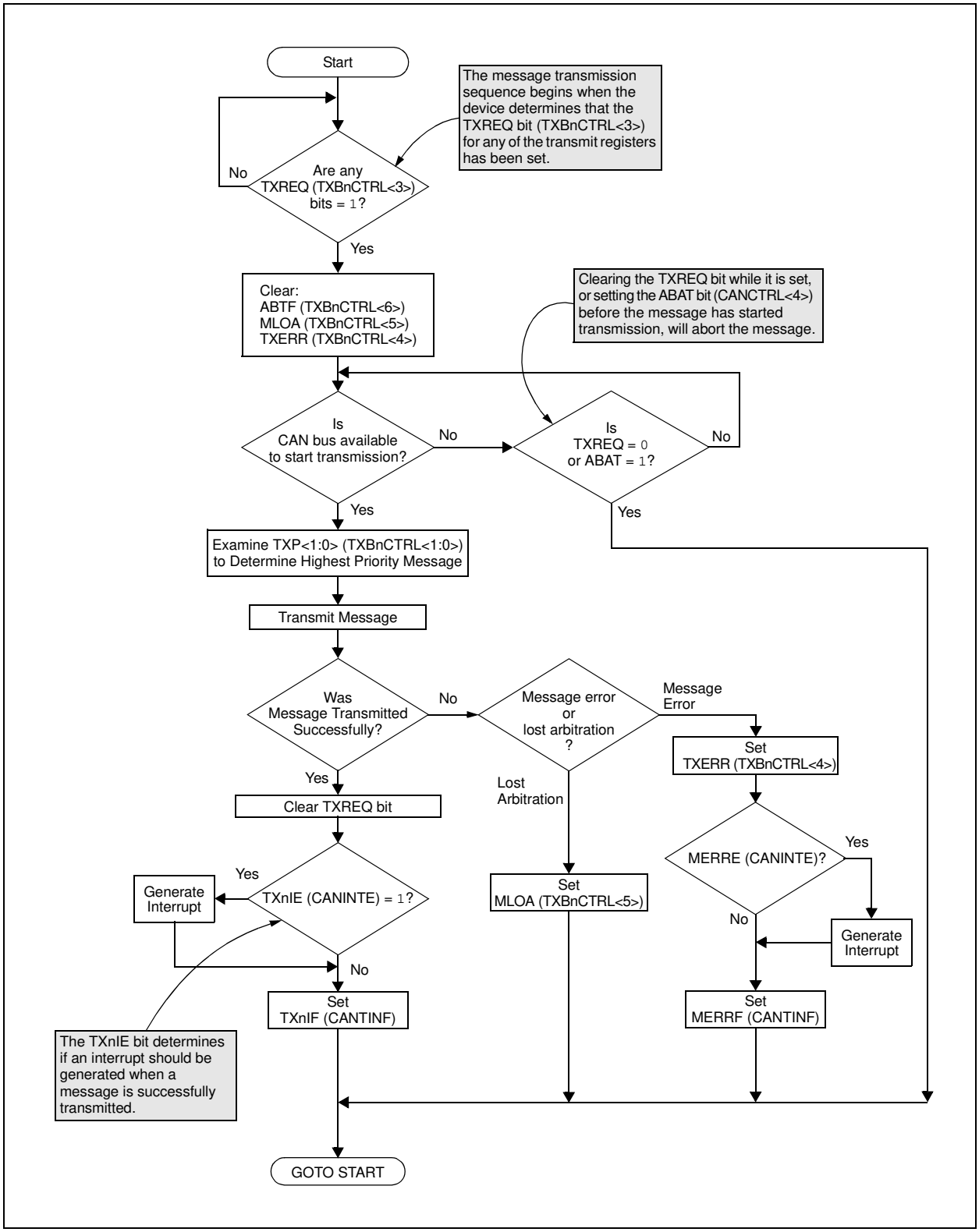
The MCU can request to abort a message in a specific message buffer by clearing the associated TXREQ bit.

In addition, all pending messages can be requested to be aborted by setting the ABAT bit (CANCTRL<4>). This bit MUST be reset (typically after the TXREQ bits have been verified to be cleared) to continue transmitting messages. The ABTF flag (TXBnCTRL<6>) will only be set if the abort was requested via the ABAT bit. Aborting a message by resetting the TXREQ bit does NOT cause the ABTF bit to be set.

Note 1: Messages that were transmitting when the abort was requested will continue to transmit. If the message does not successfully complete transmission (i.e., lost arbitration or was interrupted by an error frame), it will then be aborted.

2: When One-Shot mode is enabled, if the message is interrupted due to an error frame or loss of arbitration, the ABTF bit will set.

FIGURE 3-1: TRANSMIT MESSAGE FLOWCHART



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

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

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

bit 6 **ABTF:** Message Aborted Flag bit
- 1 = Message was aborted
- 0 = Message completed transmission successfully

bit 5 **MLOA:** Message Lost Arbitration bit
- 1 = Message lost arbitration while being sent
- 0 = Message did not lose arbitration while being sent

bit 4 **TXERR:** Transmission Error Detected bit
- 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 bit
- 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 bits
- 11 = Highest message priority
- 10 = High intermediate message priority
- 01 = Low intermediate message priority
- 00 = Lowest message priority

REGISTER 3-2: TXRTSCTRL: $\overline{\text{TXnRTS}}$ 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

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

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

bit 5 **B2RTS:** $\overline{\text{TX2RTS}}$ Pin State bit
- 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:** $\overline{\text{TX1RTS}}$ Pin State bit
- 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 bit
- 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 bit
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 bit
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 bit
1 = Pin is used to request message transmission of TXB0 buffer (on falling edge)
0 = Digital input

REGISTER 3-3: TXBnSIDH: TRANSMIT BUFFER n STANDARD IDENTIFIER REGISTER 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

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

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

REGISTER 3-4: TXBnSIDL: TRANSMIT BUFFER n STANDARD IDENTIFIER REGISTER LOW (ADDRESS: 32h, 42h, 52h)

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

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

bit 7-5 **SID<2:0>:** Standard Identifier bits

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

bit 3 **EXIDE:** Extended Identifier Enable bit
- 1 = Message will transmit Extended Identifier
- 0 = Message will transmit Standard Identifier

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

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

REGISTER 3-5: TXBnEID8: TRANSMIT BUFFER n EXTENDED IDENTIFIER 8 REGISTER 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

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

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

REGISTER 3-6: TXBnEID0: TRANSMIT BUFFER n EXTENDED IDENTIFIER 0 REGISTER 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

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

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

REGISTER 3-7: TXBnDLC: TRANSMIT BUFFER n DATA LENGTH CODE REGISTER (ADDRESS: 35h, 45h, 55h)

U-0	R/W-x	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x
—	RTR	—	—	DLC3(1)	DLC2(1)	DLC1(1)	DLC0(1)
bit 7							bit 0

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

bit 7 **Unimplemented:** Read 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 bits(1)
Sets the number of data bytes to be transmitted (0 to 8 bytes).

Note 1: It is possible to set the DLC<3:0> bits to a value greater than eight; however, only eight bytes are transmitted.

REGISTER 3-8: TXBnDm: TRANSMIT BUFFER n DATA BYTE m REGISTER (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
TXBnDm7	TXBnDm6	TXBnDm5	TXBnDm4	TXBnDm3	TXBnDm2	TXBnDm1	TXBnDm0
bit 7							bit 0

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

bit 7-0 **TXBnDm<7:0>:** Transmit Buffer n Data Field Byte m bits

4.0 MESSAGE RECEPTION

4.1 Receive Message Buffering

The MCP2515 includes two full receive buffers with multiple acceptance filters for each. There is also a separate Message Assembly Buffer (MAB) that acts as a third receive buffer (see Figure 4-2).

4.1.1 MESSAGE ASSEMBLY BUFFER

Of the three receive buffers, the MAB is always committed to receiving the next message from the bus. 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 is met.

4.1.2 RXB0 AND RXB1

The remaining two receive buffers, called RXB0 and RXB1, can receive a complete message from the protocol engine via the MAB. The MCU can access one buffer, while the other buffer is available for message reception, or for holding a previously received message.

Note:	The entire content 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.

4.1.3 RECEIVE FLAGS/INTERRUPTS

When a message is moved into either of the receive buffers, the appropriate RXnIF bit (CANINTF) is set. This bit must be cleared by the MCU 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 MCP2515 attempts to load a new message into the receive buffer.

If the RXnIE bit (CANINTE) is set, an interrupt will be generated on the $\overline{\text{INT}}$ pin to indicate that a valid message has been received. In addition, the associated $\overline{\text{RXnBF}}$ pin will drive low if configured as a receive buffer full pin. See **Section 4.4 "RX0BF and RX1BF Pins"** for details.

4.2 Receive Priority

RXB0, the higher priority buffer, has one mask and two message acceptance filters associated with it. The received message is applied to the mask and filters for RXB0 first.

RXB1 is the lower priority buffer, with one mask and four acceptance filters associated with it.

In addition to the message being applied to the RXB0 mask and filters first, the lower number of acceptance filters makes the match on RXB0 more restrictive and implies a higher priority for that buffer.

When a message is received, the RXBnCTRL<3:0> register bits will indicate the acceptance filter number that enabled reception and whether the received message is a Remote Transfer Request.

4.2.1 ROLLOVER

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.

4.2.2 RXM BITS

The RXM<1:0> bits (RXBnCTRL<6:5>) set special Receive modes. Normally, these bits are cleared 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 EXIDE bit (RFXnSIDL<3>) in the Filter n Standard Identifier Low register.

If the RXM<1:0> 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 EOF, 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.

Setting the RXM<1:0> bits to '01' or '10' is not recommended.

4.3 Start-of-Frame Signal

If enabled, the Start-of-Frame signal is generated on the SOF pin at the beginning of each CAN message detected on the RXCAN pin.

The RXCAN pin monitors an Idle bus for a recessive-to-dominant edge. If the dominant condition remains until the sample point, the MCP2515 interprets this as a SOF and a SOF pulse is generated. If the dominant condition does not remain until the sample point, the MCP2515 interprets this as a glitch on the bus and no SOF signal is generated. Figure 4-1 illustrates SOF signaling and glitch filtering.

As with One-Shot mode, one use for SOF signaling is for TTCAN-type systems. In addition, by monitoring both the RXCAN pin and the SOF pin, an MCU can detect early physical bus problems by detecting small glitches before they affect the CAN communications.

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 pins have three different configurations (Table 4-1):

1. Disabled
2. Buffer Full Interrupt
3. Digital Output

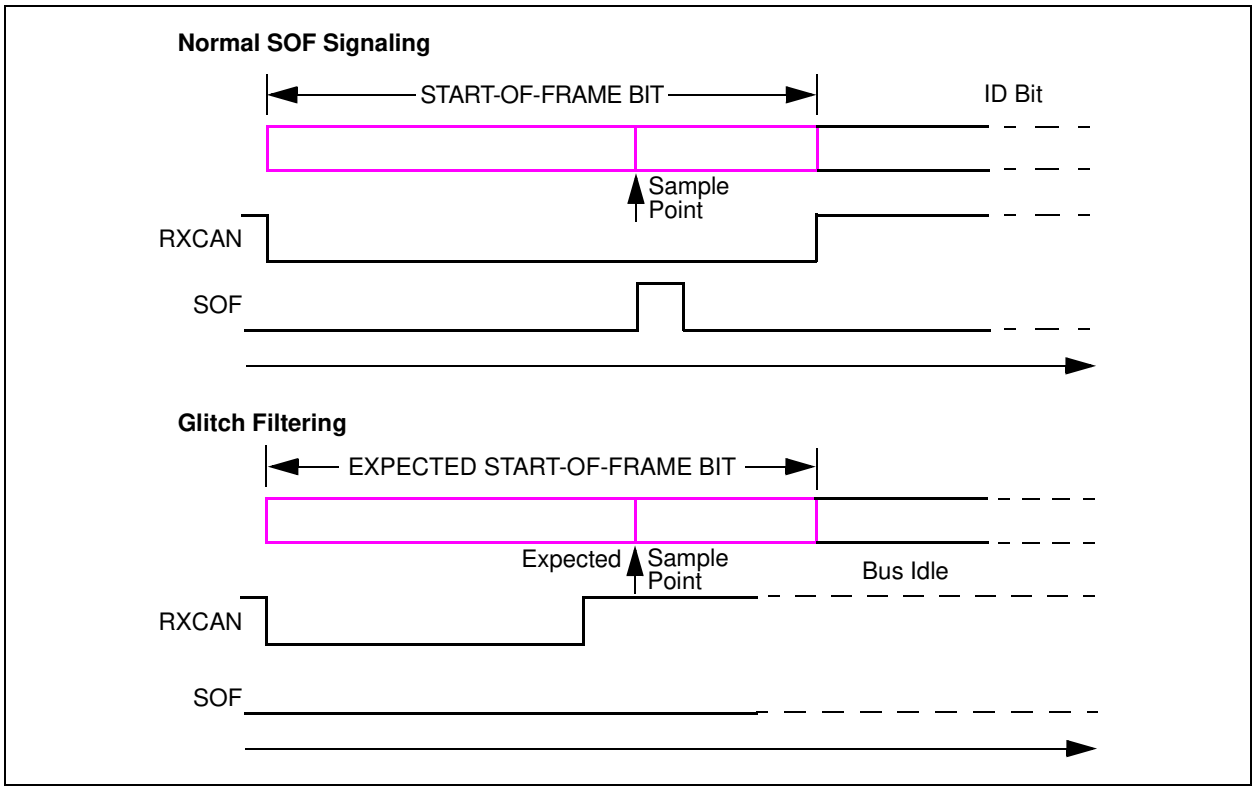
4.4.1 DISABLED

The $\overline{\text{RXnBF}}$ pins can be disabled to the high-impedance state by clearing the BnBFE bits (BFPCTRL<3:2>).

4.4.2 CONFIGURED AS BUFFER FULL

The $\overline{\text{RXnBF}}$ pins can be configured to act as either buffer full interrupt pins or as standard digital outputs. Configuration and status of these pins are available via the BFPCTRL register (Register 4-3). When set to operate in Interrupt mode, by setting the BnBFE and BnBFM bits (BFPCTRL<3:0>), these pins are active-low and are mapped to the RXnIF bit (CANINTF) 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 RXnIF bit is cleared by the MCU, the corresponding interrupt pin will go to the logic high state until the next message is loaded into the receive buffer.

FIGURE 4-1: START-OF-FRAME SIGNALING



4.4.3 CONFIGURED AS DIGITAL OUTPUT

When used as digital outputs, the BnBFM bits (BFPCTRL<1:0>) must be cleared and the BnBFE bits (BFPCTRL<3:2>) must be set for the associated buffer. In this mode, the state of the pin is controlled by the BnBFS bits (BFPCTRL<5:4>). Writing a '1' to a BnBFS bit will cause a high level to be driven on the associated buffer full pin, while 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 SPI BIT MODIFY command to prevent glitches from occurring on either of the buffer full pins.

TABLE 4-1: CONFIGURING $\overline{\text{RXnBF}}$ PINS

BnBFE	BnBFM	BnBFS	Pin Status
0	x	x	Disabled, high-impedance
1	1	x	Receive buffer interrupt
1	0	0	Digital output = 0
1	0	1	Digital output = 1

FIGURE 4-2: RECEIVE BUFFER BLOCK DIAGRAM

Note: Messages received in the MAB are initially applied to the mask and filters of RXB0. In addition, only one filter match occurs (e.g., if the message matches both RXF0 and RXF2, the match will be for RXF0 and the message will be moved into RXB0).