Since 2003, some vehicle makes and models started using a new diagnostic communication protocol called CANbus. By 2008 all vehicles sold for the United States market must use the CANbus protocol.


5.1.5 The CAN Bus
The two-wire CAN bus represents the most popular implementation of CAN. The two-wire CAN bus uses non-return-to-zero (NRZ) signaling with bit stuffing. The term NRZ means that the transmission of two successive 1 bits does not result in the signal first being lowered to zero after the first 1 bit.

The left portion of Figure 5.2 illustrates the connection of a CAN controller to a two-wire CAN bus. Note that a CAN transceiver has two connections to the bus. The first connection (CANh) is used to transmit a differential signal, while the second connection (CANI) is used to monitor the CAN bus, which also provides for the receipt of the receiver signal by the CAN controller.

The CAN controller is usually integrated on a digital signal processor (DSP) chip, which in turn is built into an electronic control unit (ECU). Thus, the CAN controller provides the mechanism whereby one ECU can communicate with another to check its status or exchange information. The left portion of Figure 5.2 illustrates the NRZ signaling method used on the CAN bus as well as the relationship between the transmit and receive signaling state and dominant and recessive signals on the bus.

Now that we have an appreciation for the term NRZ and the connection of a CAN controller to the two-wire CAN bus, let us turn our attention to the term bit stuffing. Bit stuffing prohibits the transmission of a string of six consecutive zero (000000) or six consecutive one (111111) bits by inserting an opposite bit in the data stream to prevent the transmission of six bits set to a 0 or 1. At the receiver an opposite action occurs, with the receiver removing the stuffed bit. Under CAN a bit stuffing violation in which six consecutive bits of the same type are received is considered to represent an error.

CAN uses a modified carrier sense multiple access with collision detection (CSMA/CD) method for nodes to gain access to the bus, in addition to an arbitration process. In a CAN each device listens to the bus to determine if the message flowing on the bus is the same as it is trying to transmit. If it is different, the device CAN bus CAN controller
CAN transceiver
Txd Dominant
Dominant
Recessive
Txd
Rxd
Rxd
CANh
CANh
CAN1 CAN1

Figure 5.2 The CAN two-wire physical layer and signal on the bus.
will immediately release the bus. This process ensures that one master will always
win and results in no messages lost due to a collision.

5.1.5.1 Signaling States
Under CAN there are two different signaling states, referred to as dominant (logical
0) and recessive (logical 1). These signaling states correspond to certain electrical
levels, which depend upon the physical layer used. As we will shortly note, there
are several different physical layers that can be used by CAN.
At the CAN transceiver the connection to the bus represents a wired-AND
function. This means that if just one node is driving the bus to a dominant state,
then the entire bus will be in that state regardless of the number of nodes transmitting
a recessive state.

5.1.5.2 The Physical Layer
Currently there are several different physical layers defined under the CAN specification.
The most common physical layer specification is the one defined in the ISO
11898-2 specification for a two-wire balanced signaling method. This specification
is also referred to as high-speed CAN.
Under the ISO 11898-3 specification another two-wire balanced signaling
scheme is defined, for lower bus speeds. This is a fault-tolerant specification, which
enables signaling to continue even if one bus wire should become cut or shorted to
ground or battery. This specification is referred to as low-speed CAN.
A third common physical layer is defined by the Society of Automotive Engineers
(SAE) in the J2411 specification. This specification defines the use of a singlewire-
plus-ground physical layer, which is used primarily in certain vehicles, such
as GM automobiles.

5.1.5.3 Data Transmission
Under CAN, arbitration-free transmission is used to place data on the bus. That
is, a CAN message transmitted with the highest priority will satisfy the arbitration
while nodes transmitting lower-priority messages will sense the higher priority and
back off and wait for access to the bus.
Arbitration-free transmission is supported by the use of dominant (logical 0)
and recessive (logical 1) bits. This means that if one node transmits a dominant bit
while another node transmits a recessive bit, then the dominant bit wins the arbitration. Table 5.1 indicates the bus state for two nodes transmitting as well as the value of a logical AND between the two.

Based upon the entries in the truth tables shown in Table 5.1, let us assume one node is transmitting a recessive bit (logical 1) when another node transmits a dominant bit (logical 0). The node transmitting the recessive bit (0) sees the dominant bit (which creates a voltage across the bus while the recessive bit is not asserted on the bus) and determines a collision occurred. Thus, the node transmitting a recessive bit will back off. Then, instead of transmitting, it will wait six bit durations after the end of the dominant message prior to attempting to retransmit.

During the arbitration process each transmitting node will monitor the state of the CAN bus, comparing the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted the node will stop transmitting.

The actual arbitration process commences during the transmission of the identifier field in the message frame. Each node that commences transmission at the same time places an identifier field with a dominant bit as binary 0, beginning with the high-order bit. As soon as the node ID is a larger number, which indicates a lower priority, the node will transmit a binary 1 (recessive) and observe a binary 0 (dominant), which provides the indication to back off and wait. At the end of the transmission of the identifier field all nodes except the node with the highest-priority message will have backed off, while the node with the highest priority continues its transmission. This action results in the higher-priority message gaining access to the bus, while lower-priority messages will automatically retransmit in the next bus cycle or in a subsequent bus cycle, assuming that there are other higher-priority messages waiting to gain access to the bus.

5.1.5.4 Interoperability Issues

Because different physical layers as a rule are not interoperable, the cost of CAN components, such as transceivers, cannot be amortized over a very large number because different transceivers are used with different physical layers. This is turn drives the cost of CAN upward and results in the use of LIN as a mechanism to provide a lower-cost communications capability for groups of up to 16 slave nodes.

Table 5.1 Truth Tables for Dominant/Recessive and Logical AND

<table>
<thead>
<tr>
<th>Bus State with Two Nodes Transmitting Logical AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dominant Recessive 01</td>
</tr>
<tr>
<td>Dominant Dominant 000</td>
</tr>
<tr>
<td>Recessive Dominant 101</td>
</tr>
</tbody>
</table>

5.1.5.5 Bus Speed

As previously mentioned in this section, the maximum speed of a CAN bus (ISO11898-2) is 1 Mbps, while a low-speed CAN (ISO11898-3) has a data rate up to 125 kbps. In addition, a single-wire CAN has the ability to transmit at a data rate up to approximately 50 kbps in its standard mode of operation, while its high-speed mode allows a data transfer capability of up to approximately 100 kbps. Because the type of transceiver used also governs the obtainable data rate, it is possible that
transmissions may have both an upper and lower boundary, as some transceivers cannot transmit below a certain data rate.

5.1.5.6 Cable Length
At a data rate of 1 Mbps a maximum cable length of 40 m, or 130 ft, can be supported. Because the pulse width is inversely proportional to the data rate, slowing the transmission rate widens pulses transmitted on the bus, which in turn enables the transmission distance to be extended. Thus, at a data rate of 500 kbps the maximum cable length is increased to 100 m (330 ft), while at a data rate of 250 kbps the maximum cable length is extended to 500 m (1600 ft). From a practical standpoint, the lower-speed versions of CAN are more suitable for the factory floor where extended cable lengths are required.

5.1.5.7 Bus Termination
Under the ISO 11898 CAN standard the CAN bus must be terminated. The termination of the ends of the bus is accomplished through the use of a 120-ohm resistor at each end of the bus. The use of 120-ohm resistors removes potential signal reflections at the end of the bus as well as ensures that the correct DC level flows on the bus.

Figure 5.3 illustrates the structure of a typical CAN bus. Note that each end has a 120-ohm resistor to remove signal reflections. The actual bus length will vary based upon the data rate of the bus, with higher data rates reducing the length of the bus.

Device 1
120-ohm resistor
terminator
120-ohm resistor
terminator
Device 2 • • • Device n
Figure 5.3 A typical CAN bus.

5.1.5.8 Cable and Cable Connectors
Under the ISO 11898 specification a twisted-pair cable that can be either shielded or unshielded is acceptable. Under the SAE J2411 specification a single wire is defined for use.

Although there is presently no standard defined for CAN connectors, the higher layer of the protocol stack defines a few preferred connectors. Three of those connectors are the nine-pin D-sub, five-pin Mini-C, and six-pin Deutsch.

The top portion of Figure 5.4 illustrates the pin positions and assignments for the nine-pin D-sub connector. This illustration represents a male connector viewed from the connector side or a female connector viewed from the soldering side. The power portion of Figure 5.4 contains a table indicating the pins and pin assignments of the connector.

Although the nine-pin D-sub connector is the most popularly used in a CAN, both the five-pin Mini-C and six-pin Deutch connectors are also used. The five-pin Mini-C connector resembles two concentric circles with five pins spaced within the
inner concentric circle and is compatible with both standard and extended CAN.
The six-pin Deutsch connector is primarily used for mobile hydraulic applications.

5.2 Message Frames
CAN has the ability to support the transmission of four different message types, with each message broadcast on the bus. This means that all nodes literally hear each transmission, requiring hardware to provide local filtering that enables a node to react to messages of interest to the node. The four types of messages that can flow on a CAN bus include:
- Data frame
- Remote frame
- Error frame
- Overload frame

Figure 5.4 Nine-pin DSUB connector and pin assignments: 1 = reserved; 2 CAN_L = CAN_L bus line (dominant low); 3 CAN_GND = CANZ Ground; 4 = reserved; 5 CAN_SHLD = optional CAN shield; 6 GND = optional CAN ground; 7 CAN_H = CAN_H bus line (dominant high); 8 = reserved (error line); 9 CAN_V+ = optional power.

5.2.1 Data Frame
The CAN data frame represents the most common type of message transmitted on the CAN bus. The first version of CAN, which is defined by the ISO 11519 specification, uses an 11-bit identifier field that, when combined with a one-bit remote transmission request (RTR) field, is used to determine the priority of messages when two or more nodes are contending for access to the common bus. This version of CAN operates at data rates up to 125 kbps and is referred to as standard CAN. A second CAN data frame uses a 29-bit identifier formed by adding an 18-bit identifier field to the standard CAN frame as well as incorporating three modifications to the frame, which we will shortly discuss. This type of frame is referred to as an extended frame and can operate at data rates up to 1 Mbps. For an extended CAN data frame the arbitration field, which is employed to determine the priority of messages when two or more nodes contend for access to the bus, consists of a 29-bit identifier field formed by separate 11-bit and 18-bit identifier fields and the RTR bit. Now that we have a basic appreciation for the two types of data frames, let us examine their composition in detail.

5.2.1.1 Standard Data Frame
Figure 5.5 illustrates the fields in the standard CAN data frame. Both the low-speed CAN, defined by the ISO 11519 specification, and CAN 2.0A, defined by the ISO 11898 specification, are compatible with the use of an 11-bit identifier field. The primary difference between the two ISO specifications is the fact that the original standard CAN operates at 125 kbps while CAN 2.0A operates at 1 Mbps. In examining both Figure 5.5, which illustrates the standard CAN data frame format, and Figure 5.6, which shows the extended data frame format, you will note the absence of an address field. Because CAN messages are broadcast on the bus,
there is no need for an address field. Instead, CAN messages can be considered to be content addressed because the contents of a message determine if a node acts upon a message.

Another item worth noting is the fact that the presence of an ACK bit does not indicate that any of the intended nodes have received the message. This bit can be set by any controller that was able to correctly receive the message, by sending an ACK bit at the end of the message. Thus, the ACK bit only informs us that one or more nodes on the bus correctly received the message.

5.2.1.2 Extended Data Frame

The extended CAN data frame, as previously noted in this chapter, uses a 29-bit identifier. To extend the identifier, the extended CAN frame added an 18-bit identifier field, which is separated from the original standard CAN 11-bit identifier field by two fields, a substitute remote request (SRR) field and an identifier extension (IDE) field.

Figure 5.6 illustrates the format of the extended CAN message frame. In comparing Figure 5.4 to Figure 5.6, note that other than the use of an 18-bit identifier field to extend the identifier to 29 bits, the extended CAN data frame only differs from the standard CAN data frame by the addition of three fields:

11-bit identifier 18-bit
r0 r1 0 ... 8 data bytes
SOF SRR
IDE
CRC
ACK
EOF
IFS

Figure 5.6 The extended CAN message frame.

11-bit identifier r0 0 ... 8 data bytes
SOF
RTR
IDE
CRC
ACK
EOF
IFS

Figure 5.5 Standard CAN data frame format. SOF, a bit that marks the start of a frame and is used to synchronize nodes on a bus after being idle; identifier, an 11-bit identifier that establishes the priority of a message, where a lower value indicates a higher priority; RTR, the remote transmission request bit is set when information is required from another node (although all nodes on the bus receive the request, the identifier determines the node that responds); IDE, the single identifier extension bit is set to define a standard CAN identifier without an extension; R0, a reserved bit for future use; DLC, a 4-bit data
length code that indicates the number of bytes of data transmitted; data, 0 to 64 bits (8 bytes); CRC, a 15-bit cyclic redundancy check containing the checksum (number of bits transmitted) of the preceding application data for error detection (in actuality the CRC field consists of a 15-bit CRC and a recessive delimiter bit that indicates the end of the field); ACK, each node that receives an accurate message overwrites this bit position with a dominant bit, which indicates the message was received error-free (if a receiving node detects an error, the message is discarded and the sending node repeats the message; the ACK field is two bits in length, with the first bit used for acknowledgment and the second functioning as a delimiter); EOF, a seven-bit end-of-frame (EOF) field marks the end of a CAN message; IFS, a seven-bit inter-frame separator (IFS) represents the amount of time required by a controller to move a correctly received frame into its message buffer area.

SRR — Substitute remote request bit, which replaces the RTR bit in the standard message location as a placeholder in the extended frame
IDE — Identifier extension (IDE) bit, which indicates that an 18-bit extension identifier follows
R1 — An additional reserved bit

5.2.1.3 Arbitration
For both the standard and extended CAN frames the arbitration field that is used to determine the priority of a message when two or more nodes contend for use of the bus can be considered to represent a pseudo-field. This field under standard CAN contains an 11-bit identifier and the RTR bit, which is dominant for data frames. Under extended CAN the arbitration field consists of a 29-bit identifier, two recessive bits (SRR and IDE), and the RTR bit.

5.2.1.4 Bit Stuffing
For both standard and extended CAN frames bit stuffing results in the insertion of a bit of opposite polarity after a sequence of five bits of the same polarity occurs. Bit stuffing covers both standard and extended frames from the start-of-frame bit field through the 15-bit cyclic redundancy code field.

5.2.2 Remote Frame
A third type of message that can be transmitted on a CAN bus is the remote frame. The remote frame is similar to the standard and extended CAN data frames. However, there are two key differences between the remote frame and each type of data frame. First, the remote frame has no data field. Second, the remote frame is explicitly marked by the RTR bit being set recessive.

5.2.2.1 Operation
Remote frames can be used to invoke a request–response type of bus traffic. For example, if Node A transmits a remote frame with its arbitration field set to a value of 246, then a node that determines the request frame requires a response would respond with a data frame with its arbitration field similarly set to a value of 246. Unlike data frames that commonly flow on a CAN bus, remote frames are not
commonly used. However, when used, the data length code field must be set to the length of the expected response. If not, arbitration will not work.

5.2.3 Error Frame
A fourth type of frame supported by CAN is the error frame. This frame is transmitted by any node detecting an error. In actuality, the error frame represents a special message that violates the rules of a CAN message. The error frame is transmitted when a node detects an error in a message, causing all other nodes in the network to similarly transmit an error frame. The original node that transmitted the error frame automatically retransmits the message. Through the use of error counters in the CAN controller, which will be reviewed in the next section in this chapter, a node is prevented from continuously transmitting error frames, which in effect would lock up the bus.

The error frame consists of two fields. The first field is the error flags, which is created by the superposition of error flags contributed by different nodes on the bus. There are two types of error flags: active and passive. An active error flag is transmitted by a node that detects an error on the network that is in the "error active" error state. In comparison, a passive error flag is transmitted by a node that detects an active error frame on the network that is in the "error passive" error state.

5.2.4 Overload Frame
The fifth type of frame that can flow on the CAN bus is the overload frame. This frame is transmitted by a node that becomes too busy to process additional data. Thus, the purpose of this frame is to provide for an additional delay between messages.

5.3 Error Handling
In concluding our examination of the operation of the Controller Area Network we will turn our attention to one of the more important aspects of the technology: the manner by which error handling occurs. However, prior to doing so, let us first briefly review how conventional communications technology detects and corrects errors, as this will provide a frame of reference for comparing CAN error handling to common communications error handling.

5.3.1 Communications Error Handling
In a modern communications environment error handling occurs through either the use of parity, when bytes are transmitted independently of one another, or the use of a checksum, when bytes are grouped into a block for transmission.

5.3.2 Parity Checking
Under parity checking an extra bit, referred to as a parity bit, is added to each byte to be transmitted. Parity bit checking can be either odd or even. Under even parity bit checking the parity bit is set to a binary 0 if the number of set bits in the byte to be transmitted is even. If the number of set bits is odd, then the parity bit is set to a binary 1 so that the sum of all set bits is even. Under odd parity bit checking the parity bit is set to a binary 1 if the number of bits set in the byte are an even number and to a binary 0 if the number of bits set in the byte are an odd number.
Under parity checking only a single bit error can be detected. In addition, there is no easy way to correct a byte with a bit error other than visually or by retransmission of an entire document. Due to these problems, most error detection and correction methods evolved through the blocking of bytes and the addition of a checksum to the block that is computed based upon the use of a predefined algorithm.

5.3.3 Block Checking
Under a communications block checking method a fixed number of bytes are used to generate a block. For example, one common communications protocol that employs block checking is the Xmodem protocol. Under the Xmodem protocol 128 bytes are used to form a block. If the last block is only partially filled with data, then the remainder of the block is filled with pad characters (ASCII 127) until the block is filled with 128 characters.

Under block checking an algorithm is applied to each block to generate a checksum that is appended to the block. Thus, the block and its checksum are transmitted. At the receiver the same algorithm is applied to the received data block and a locally generated checksum is computed. The locally generated checksum is then compared to the transmitted checksum. If they match, the data block is assumed to have been received error-free. Then the checksum is removed and the block is sent from the receiver’s buffer for processing on the local computer. In addition, the receiver transmits an acknowledgment to the sender, which informs the sender that it is okay to send the next data block. If the two checksums do not match, one or more bit errors are assumed to have occurred. Thus, the receiver will transmit a negative acknowledgment to the sender and place the received data block and checksum into the great bit bucket in the sky. The negative acknowledgment serves to inform the sender to resend the data block to include its checksum. Thus, errors are corrected by retransmission.

Although the use of a checksum lowers the probability of an undetected error, such errors can occur when the algorithm used to create the checksum is relatively simple. To reduce the probability of an undetected error, modern communications systems use a polynomial approach to error checking. That is, the bytes in the data block to be transmitted are assumed to represent a long polynomial, which is divided by a fixed polynomial. The resulting quotient is discarded while the remainder becomes the checksum. However, when a polynomial approach is used, the remainder is referred to as a cyclic redundancy check (CRC), which is placed into a CRC field.

Now that we have an appreciation for the manner by which conventional communications systems perform error handling, let us turn our attention to CAN error handling.

5.3.4 CAN Error Handling
Similar to conventional communications systems, an error handling capability is included in the CAN protocol. Under CAN there are five ways that an error can be detected. Two ways operate at the bit level, while the other three operate at the message level. In general, detecting errors in a message appearing on the CAN bus will result in the controller that detected the error transmitting an error flag. The
error flag informs other controllers on the bus to discard the current message, in effect eliminating bus traffic. Then, the originating transmitter will retransmit the previously erroneous message. Thus, the error flag can be thought of as similar to a negative acknowledgment, while errors are corrected via retransmission.

5.3.5 Node Removal

One of the more interesting aspects of CAN error handling is the ability of a node to remove itself from the CAN bus under certain conditions. To obtain the ability to determine if a node should leave the bus, each node maintains two error counters. One error counter increments when a transmit error occurs and logically has the name transmit error counter. The second error counter is incremented when a receive error occurs and has the name receive error counter. Because it is logical to expect that a transmitter detecting an error increments its transmit error counter faster than the listening nodes on the bus will increment their receive error counter, because there is a high probability that the transmitter caused a detected error, the transmit error counter value can be used as a threshold for action. That is, once the transmit error counter value reaches a predefined value, the node associated with the counter will first go into an error passive state. When in an error passive state the node will not actively transmit an error flag when an error occurs. Next, the node will then go into a “bus off” state, which means that the node will not participate in any bus traffic.

5.3.6 Error Detection Methods

As mentioned earlier in this section, the CAN protocol defines five methods whereby errors can be detected. Those methods can be categorized as error detection at the bit level and at the message level. In this section we will turn our attention to the manner by which the CAN protocol detects errors. Those methods used by CAN for error detection are summarized in Table 5.2.

5.3.6.1 Bit Monitoring

Bit monitoring is one of two bit-level error detection methods used by the CAN protocol. Under bit monitoring each transmitter on the CAN bus “reads” the transmitted signal level. If the bit level differs from the one transmitted, the node connected to the bus generates a bit error signal.

5.3.6.2 Bit Stuffing

The second error condition to occur at the bit level results from the bit stuffing process. As discussed earlier in this chapter, when five consecutive bits of the same level (0s or 1s) have been transmitted by a node, the node will add a sixth bit of the opposite level to the transmitted bit stream. This process is similar to the zero insertion method used by the High-level Data-Link Control (HDLC) protocol. That method prevents a sequence of six consecutive binary 1 bits from appearing between two flags that define the beginning and ending of a communications frame. When five consecutive 1 bits occur in any part of a frame other than the beginning and ending flag, the sending station inserts an extra 0 bit. When the receiving station detects five 1 bits followed by a 0 bit, it will remove the previously
inserted 0 bit, restoring the bit stream to its original value. Thus, under HDLC a false frame is precluded from occurring due to the zero insertion process. The bit stuffing method utilized by the CAN protocol is used as a mechanism to prevent excessive DC voltage bus buildup. That is, under CAN data is transmitted using non-return-to-zero (NRZ) coding. This coding method means that a sequence of binary 1s results in a high voltage level for the bit duration of all bits, while a sequence of binary 0s would result in no voltage for the bit duration of the sequence of zero bits. Because a long string of binary 1s could result in DC voltage buildup, while a long string of binary 0s could result in a loss of synchronization, bit stuffing under CAN treats sequences of consecutive bits of each polarity the same. This explains why a sixth bit of the opposite polarity is added to the outgoing bit stream when five consecutive bits of 1s or 0s are transmitted by a node. Because bit stuffing changes any long sequence of binary 0s or binary 1s, if more than five consecutive bits of the same polarity occur on a bus, this represents an error condition. The error condition that is signaled is referred to as a stuff error.

5.3.6.3 Frame Check
The frame check is one of three message errors that can occur under the CAN protocol. Because the CAN message has a number of fixed fields that have a range of predefined values, a single value, or a computed value, it becomes possible to check certain CAN message fields. For example, the CRC delimiter, ACK delimiter, and end-of-frame fields have values that can be easily checked. If the CAN controller detects an invalid value in one or more of these fixed fields it will initiate a form error signal.

5.3.6.4 Acknowledgment Check
A second error message that can occur at the message level is referred to as an acknowledgment error. As a review, all nodes that correctly receive a message regardless of the message destination under the CAN protocol are expected to send a dominant level in the acknowledgment slot in the message, while the transmitter places a recessive level in the slot. If the transmitter does not detect a dominant level in the acknowledgment slot, then an acknowledgment error is signaled.

5.3.6.5 Cyclic Redundancy Check
Each CAN message includes a 15-bit cyclic redundancy check (CRC). This CRC is similar to a checksum, but computed by treating the message as a long polynomial, which is then divided by a fixed polynomial. This results in a quotient and remainder,
with the quotient discarded and the remainder becoming the 15-bit CRC. Each node performs a similar computation on the transmitted message, using the same fixed polynomial. If a node’s computed CRC does not match the transmitted CRC, a CRC error will be signaled.

5.3.7 CAN Controller Operations
Previously, we noted that a CAN controller can increment two counters, one of which corresponds to recognition that a transmitter error occurred, while the second counter is incremented in recognition that a receive error occurred. In actuality, a CAN controller has its mode of operation controlled by the two error counters. In this section we will examine the states that a controller can be in and how the values of the two error counters are used to move the controller from one state to another.

5.3.7.1 Controller States
A CAN controller can be placed into one of three defined error states: error active, error passive, and bus off. The error active state enables messages to be transmitted and received. Thus, this state represents the normal operating mode of a controller. When an error is detected, an error flag is transmitted by the controller.

The second CAN controller error state is the error passive state. A node enters the error passive state when a controller experiences frequent problems when transmitting and receiving messages. Although the controller can transmit and receive messages in an error passive state, it will transmit an error flag when it detects an error when receiving data.

The third CAN controller state is bus off. A controller enters this state if it experiences significant problems when transmitting messages. Once the controller enters the bus off state it cannot transmit or receive messages until it is reset by the host microcontroller or processor.

5.3.7.2 Mode Control
The actual mode of operation of a CAN controller is determined by the contents of the transmit error counter and the receive error counter. The CAN controller will be in an error active mode when both the transmit error counter and receive error counter contents are each less than or equal to 127. If the transmit error counter value is greater than 127 but less than or equal to 255, the CAN controller will be placed into an error passive mode of operation. Only if the contents of the transmit error counter exceed 255 will the CAN controller be placed into a bus off state of operation. When this situation occurs, the CAN controller must be reset by the host microcontroller or processor to be able to resume an operational capability.

5.3.7.3 Counter Updating
Because the contents of the transmit error counter and receive error counters are the mechanism by which a CAN controller resides in a particular state, let us discuss how those counters are updated.

5.3.7.4 Receiver Error Counter
When a receiver detects an error it will normally increment the value of the counter
by 1. There are two exceptions to this. One exception occurs when the detected error was a bit error when an error flag or an overload flag was transmitted. If the receiver detects a dominant bit as the first bit after sending an error flag, its receive error counter will be increased by 8.

The second exception occurs when a node detects 14 consecutive dominant bits after sending an active error flag or an overload flag, or after detecting 8 consecutive dominant bits following a passive error flag and after each sequence of 8 additional consecutive dominant bits. When any of these conditions occurs, every receiver on the bus will increment its receiver error counter value by 8.

5.3.7.5 Transmit Error Counter

The operational setting of the transmit error counter is slightly different from that of the receive error counter. Those differences include decrementing the counter value by 1 after the successful reception of a message unless the value was already 0. In addition, the counter value will also be decreased by 1 if it is between 1 and 127 upon the successful reception of a message, to include the successful sending of the acknowledgment bit. In this situation, if the counter value was 0, it will remain at 0, while a counter value greater than 127 will be set to a value between 119 and 127. Another key series of differences between the transmitter error counter and the receive error counter are two situations that cause the transmit error counter to be incremented by 8:

- When a transmitter sends an error flag
- When a transmitter detects a bit error while sending either an active error flag or an overload flag

For both of the above situations the transmitter error counter value will be incremented by 8.

5.3.7.6 Error Signaling

Previously we noted the different types of bit and message errors. However, in doing so we deferred until now the details concerning the manner by which different error signals are formed. Thus, let us turn our attention to this important topic.

When a node detects an error it will place an error flag on the bus as a mechanism to prevent other nodes from accepting the erroneous message. The active error flag consists of a sequence of six low or dominant bits. This sequence of six consecutive low bits represents an intentional bit stuffing violation, which will be detected by all other nodes on the bus. Each of the nodes will respond with its own error flag. Once this occurs, the nodes that need to transmit, to include the node that originated the active error flag, will begin their transmissions. This will result in the occurrence of the CAN arbitration process, where the message with the highest priority wins the arbitration process and obtains the ability to transmit its message.

When the CAN controller is in an error passive mode, the error frame will be in the reverse state of an active error flag. That is, it will consist of six passive or high bits. Because the error flag now consists of passive bits, the bus is not affected. Thus, if no other nodes on the bus detect an error, the message will reach its destination without interruption. Note that the error flag is used when a node has recognized a receiving problem that does not require the bus to be affected. Because error
handling is automatically performed by the CAN controller, there is no need for the host microcontroller to perform any error handling operations. Thus, the error handling performed by the CAN controller enables the microcontroller to perform other functions.