

An Introduction to CAN

by Peter Bagschik

Advanced CAN Technology
realized by I+ME ACTIA

The CAN (Controller Area Network) protocol was developed in Europe for the use in passenger cars. Through the successful use of CAN in automotive and industrial applications such as control devices, sensors and actuators, CAN found its way to the US and other parts of the world. Internationally standardized under ISO 11898.

CAN in Cars

With the growing demand for greater safety, comfort, convenience and compliance requirements for improved pollution control and reduced fuel consumption, the car industry has developed a multitude of electronic systems (e.g. ABS, EMS, traction control, air-bags, central door locking, powered seat...). The complexity of these control systems and the need to exchange data among them required more and more hard-wired, dedicated signal lines. CAN offers the complete solution. Utilizing CAN, controllers, sensors, and actuators communicate with each other, in real-time, at speed of up to 1 MBit/s, over a two wire serial data bus.

CAN in Industrial Applications

The benefits of reduced cost and improved reliability that the car industry gains by using CAN are now used by manufacturers of a wide range of products. Industrial examples are marine control and navigation systems, elevator control systems, agricultural machinery, factory automation, photo-copiers, medical systems, textile production machinery.

CAN Networks...

- Are cost effective to design and implement
- Will continue to operate in harsh environments
- Are easy to configure and modify
- Automatically detect data transmission errors
- Provide an environment that enables centralized fault diagnosis - during design or while in service

Associated specifications such as CAN-based CAL, CANopen, DeviceNet, CAN Kingdom, SDS and J1939 are some of the enhancements to the original standard.

1 CAN Hardware/Software Concepts

A typical networked system with distributed controllers is shown in Fig. 0.1. In this case a microcontroller is the local intelligence that controls parts of the process: A corresponding algorithm analyzes the sensor signals of the process - e.g. an optical pattern detection of a workpiece - and starts the corresponding actuator functions for the next part of the process - e.g. turning around the workpiece with a certain angle.

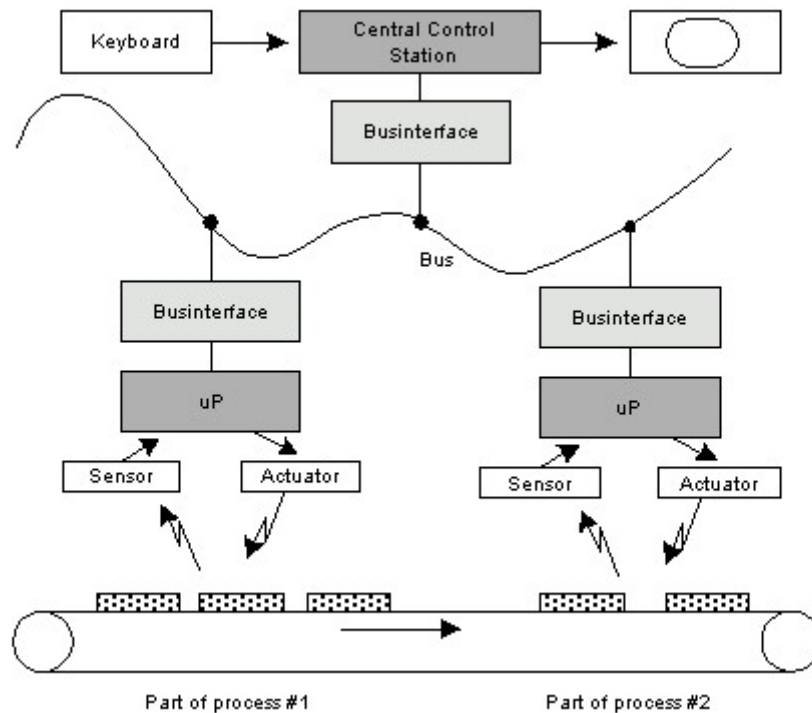


Fig. 0.1 Networked System with Distributed Controllers

The local regulator communicates with the rest of the distributed regulators and with a higher controller. The higher controller is used for setting up the system and/or to acquire status and statistical data from the process.

The communication between the parts of the process will be done using the CAN bus. Such a serial connection is recommended in case of a loosely coupled system this means a communication occurs not very often. In this case a not flexible and expensive special wiring is not necessary. The system can be modified in terms of network nodes without changing the wiring. An additional advantage using a network with a logical addressable or content addressable structure is the following:

The acquired data of the workpiece that are distributed from controller #1 in the network can be received from controller #2 and the higher controller at the same time. It is not necessary to physically address the information 2 times. The algorithm is the following: The message with the information "workpiece data" is transmitted on the bus with a special identifier. This identifier is the key in such a way that every network node is able to detect the message as "workpiece data". So it is possible that every network node who is interested in this information receives the message and gets the result.

2 CAN Specification 2.0A and 2.0B

The ISO/OSI Layer 1 and Layer 2 are described in the international standard ISO 11519-2 for low speed applications and ISO 11898 for high speed applications. In addition to the strictly form of the ISO/OSI description the CAN specification 2.0A and 2.0B are available which are more oriented to the requirements of manufacturers of CAN controllers. The difference between CAN 2.0A and CAN 2.0B is basically located in the format of the message header especially the identifier. The CAN specification 2.0A defines CAN systems with a standard 11 bit identifier (Standard CAN). CAN 2.0B specifies so called extended frames with 29 bit identifier (Extended CAN). In the next chapters the resulting message types and the resulting system features are described.

CAN Messages

Every CAN Message consists of a certain number of bits that are divided into fields. These fields have different meanings like End Of Frame, CRC Checksum, Data Field or Arbitration Field. The meaning of the Arbitration Field is on the one hand the priority of the message on the other hand it is the logical address of the information. This logical address consists of 11 bits for CAN 2.0A messages and of 29 bits for CAN 2.0B messages. As described in the ISO 11519-2 and ISO 11898 there are allowed 2032 different logical addresses for CAN 2.0A, this means there are existing 2032 different communication objects (messages) in Standard CAN. In Extended CAN networks there are allowed

229 (= 536,870,912) different messages. The exact construction of CAN messages can be seen in Fig. 0.1.

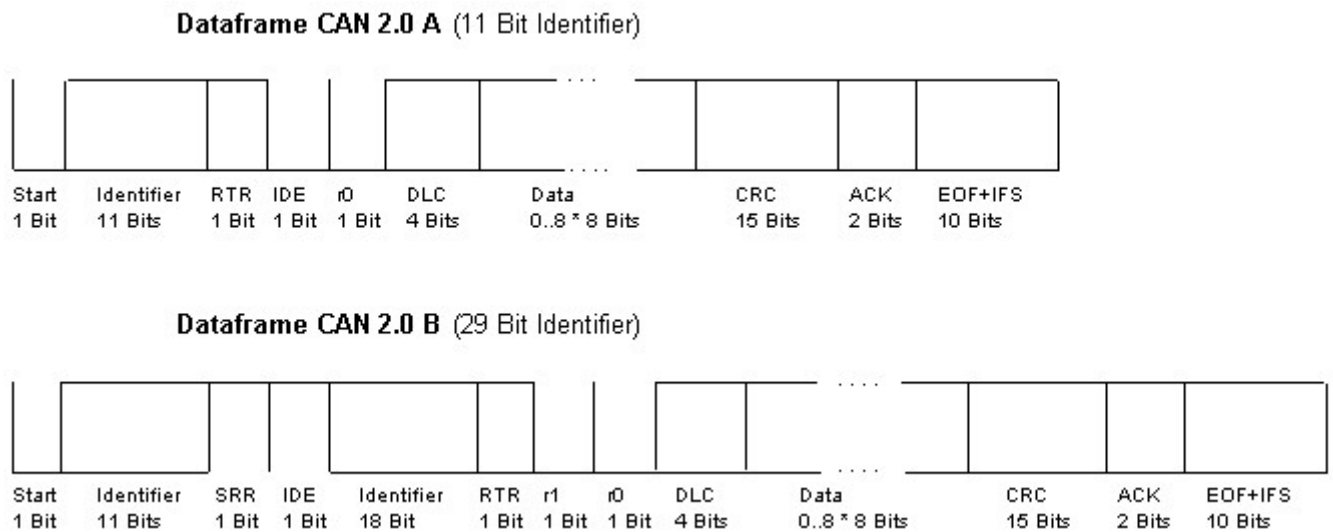


Fig. 0.1 CAN Data Messages

The meaning of the bitfields for Standard CAN messages are the following:

- · Start Bit = 1 Bit = low (dominant)
Marks the start of a message. After a idle-time (bus not used) the falling edge of the Start Bit is used for a synchronization of the different network nodes.
- · Identifier = 11 Bits
Logical address and priority of the message; the less the value the higher is the priority (0 -> highest priority)
- · RTR Bit = 1 Bit
The RTR Bit (Remote Transmission Request) is used by a receiver to request a remote transmitter to send his information. If this bit is set to 1 (= recessive) the frame contains no data field even if the data length code defines something other. In this case all the other network nodes can check whether there is the corresponding transmitter defined or not. The request and the possible answer are 2 complete different frames on the bus. This means the answer can be delayed due to messages with higher priorities.
- Control Field = 6 Bits
The first bit in the Control Field is the IDE Bit (Identifier Extension). If this bit is transmitted as logical 0 (dominant) the meaning is that there are no more identifier bits are sent (Standard CAN Frame). Bit r0 is reserved. The next four bits contain the data length code for the following data field.
- Data Field = 0 to 64 Bits (0 to 8 Bytes)
Contains the application data of the message.
- CRC Field = 16 Bits (including CRC Delimiter = high (recessive))
Contains a checksum for the preceding bits of the message. The CRC checksum will be used for error detection only. It is not used for error correction. The Hamming Distance of this CRC code is 6. With this it is possible to detect up to 6 single bit errors which are scattered about the message or so called burst errors up to a length of 15 bits.
- ACK Field = 2 Bits (including ACK Delimiter = high (recessive))
Every active network node that detects the message to be correct on the bus overwrites the recessive level (driven by the transmitter of the message) by a dominant level in the ACK Slot. This bit is also read back by the transmitter of the message. If he reads back no dominant level in this ACK Slot this treated as an error.
Note: If the transmitter detects an acknowledgment he can not be sure that the message is received by the corresponding applications. The only thing he can be sure is that the message was correct transmitted (coded) on the CAN bus.
- EOF Field = 7 Bits = high (recessive)
The EOF (End Of Frame) is marked by coding violation. The Bit Stuffing rule of the coding technique is violated. Under normal circumstances after the 5th bit with equal polarity an additional bit with reverse polarity is stuffed into the bit stream. This Bit Stuffing is switched off while the EOF is active. The EOF marks the end of a CAN frame.

- IFS = 7 Bits = high (recessive)
The IFS (Inter Frame Space) marks the space of time that is necessary for the CAN controller to transfer a correct received frame from the bus handler to the corresponding place in the message buffer area.
- Idle \geq 0 Bit = high (recessive)
Bus is not used. Every bus node is allowed to start a message transfer.

The Extended CAN messages differ from Standard CAN messages in the following positions:

- SRR = 1 Bit = high (recessive)
The SRR (Substitute Remote Request Bit) replaces the RTR Bit that would be sent in a Standard CAN message. There is no further meaning.
- IDE = 1 Bit = high (recessive)
In Extended CAN messages the IDE Bit (Identifier Extension) is set to high to indicate that there will follow more identifier bits.
- Control Field = 6 Bits
The first two bits in the control field of an Extended CAN frame are reserved (r0 and r1). The rest of this field (and the following parts of the message) have the same meaning than in Standard CAN messages.

Valid data length code values are 0..8. Data length codes greater than 8 are not allowed according to the CAN specifications.

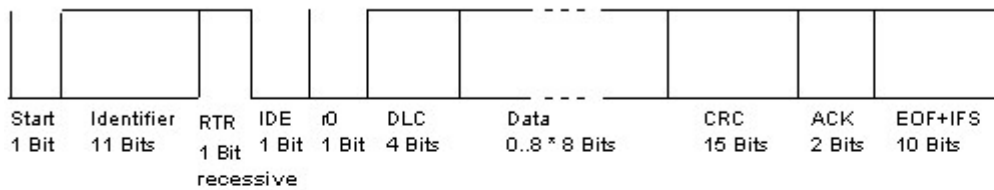
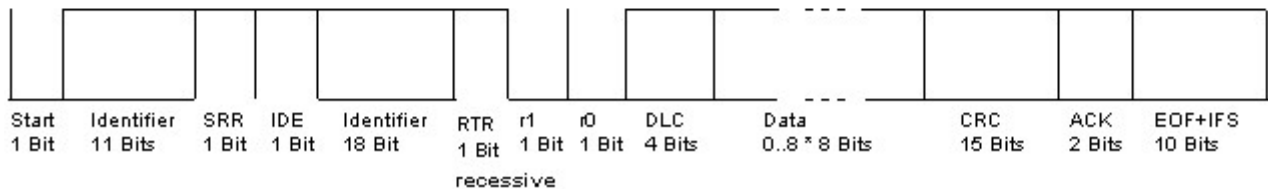
It is possible to use Standard and Extended CAN messages in one network. To do this it is necessary to use only CAN controllers with specification CAN 2.0B or CAN controllers which are so called "CAN 2.0B passive". This means a "CAN 2.0B passive" controller is able to detect Extended CAN messages by evaluating the IDE bit. This type of CAN controller ignores Extended CAN messages. A CAN controller without this feature would react with error flags.

CAN controllers according to CAN specification 2.0B are able to work with Standard CAN and Extended CAN messages at the same time. The CAN messages shown in Fig. 0.1 are normal transmit messages containing application data. Under normal circumstances this kind of message is initiated by the transmitter. In CAN networks it is possible that a message transfer is initiated by a corresponding receiver using a RTR message (Remote Transmission Request).

Every network node receives a RTR frame. With the identifier he is able to detect whether he is the corresponding transmitter of the information or not. This evaluation can either be done by the application (Basic CAN feature) or automatically by the CAN controller (Full CAN feature). The request and the possible answer are 2 complete different frames on the bus. This means the answer can be delayed due to messages with higher priorities in addition to the possible delay by the application. An advantage of this feature is that the answer (message by the transmitter containing the application data) is not only received by the requesting receiver but also by possible other receivers which are interested in this message. So this means the data consistency in the network (the distributed application) is guaranteed.

The structure of a RTR messages can be seen in Fig. 0.2.

The RTR feature can be used when a network node was temporarily off-line and gets in contact to the network again. In this case it is not necessary to wait until the transmitters have sent their data fields but he can update his receiver data fields by scanning all necessary messages.

Remoteframe CAN 2.0 A(11 Bit Identifier)**Remoteframe CAN 2.0 B(29 Bit Identifier)****Fig. 0.2 CAN RTR Messages**Frame Coding

The frame segments Start Bit, Arbitration Field (including Identifier and RTR Bit), Control Field and CRC sequence are coded with a method of bit stuffing. Whenever a transmitter detects 5 consecutive bits (including stuff bits) of identical value in the bitstream to be transmitted, it automatically inserts a complementary bit in the bitstream actually transmitted. The remaining bit fields of the data frame or remote frame (CRC delimiter, ACK field and EOF) are fixed form and are not stuffed. The Error Frame and the Overload Frame are of fixed form as well, and are not coded with bit stuffing. The bitstream is coded according to the NRZ (Non-Return-to-Zero) method. This means that generated bit level is constant during the total bit time.

Arbitration

The bus access mechanism of CAN is implemented as a non-destructive, bitwise arbitration. Non-destructive means that the winner of the arbitration - the message with the higher priority - must not restart the message from the beginning. For this feature the physical busdriver must be realized in certain way: The logical levels 0 and 1 on the bus are sent as dominant and recessive bits. It must be possible that a transmitted logical 1 is overwritten by a logical 0. For implementations of the physical bus drivers refer to chapter 5. The access technique to the bus in multimaster networks is very important concerning the performance, the delay for message transfers this means the real-time capability of a networked system. A networked system with CAN has a positive behavior concerning the real-time capability: On a collision during a bus access the most important competitor is automatically selected. In that case a transmitter detects the bus busy, the transmission request will be delayed up to the end of the actual transmission on the bus.

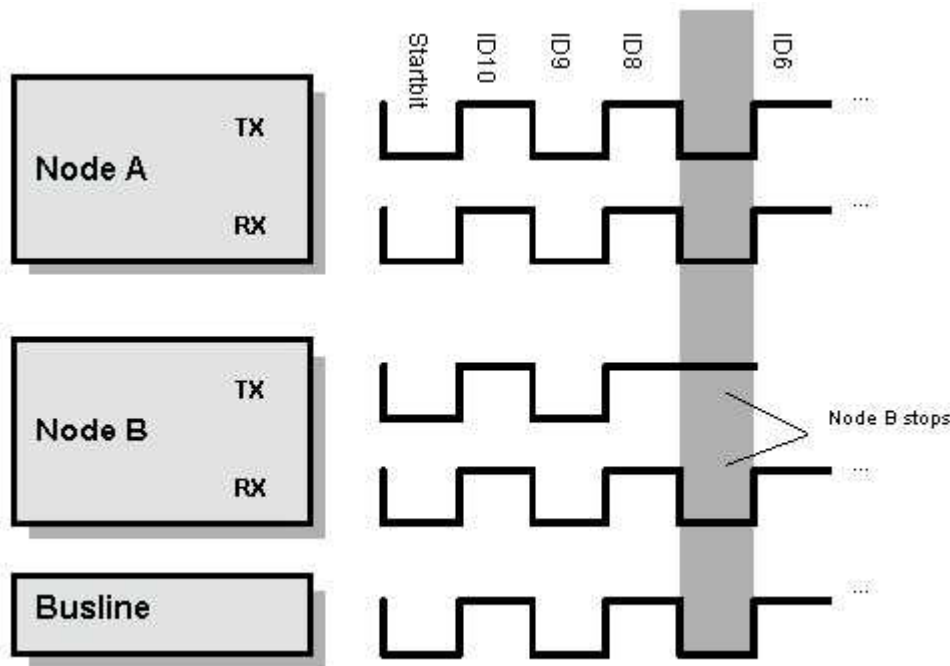


Fig. 0.3 CAN Arbitration

The arbitration phase will be explained with an example (see Fig. 0.3):

Two bus nodes have got a transmission request. The bus access method is CSMA/CD+AMP (Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority). According to this algorithm both network nodes wait until the bus is free (Carrier Sense). In that case the bus is free both nodes transmit their dominant start bit (Multiple Access). Every bus node reads back bit by bit from the bus during the complete message and compares the transmitted value with the received value. As long as the bits are identical from both transmitters nothing happens. The first time there was a difference - in this example the 7. bit of the message - the arbitration process takes place: Node A transmits a dominant level, node B transmits a recessive level. The recessive level will be overwritten by the dominant level. This is detected by node B because the transmitted value is not equal to the received value (Collision Detection). At this point of time node B has lost the arbitration, stops the transmission of any further bit immediately and switches to receive mode, because the message that has won the arbitration must possibly be processed by this node (Arbitration on Message Priority).

If this happens outside the start bit, the arbitration field and the ACK slot it is treated as a bit error and will be handled by the error management of the CAN controller.

Exception Handling

In addition to the bit errors further errors are detected and handled by the error management unit of the CAN controller. This error management unit is part of the Data-Link-Layer and specified in the international standard. Due to this any CAN controller has to work in same way concerning the error management.

One main of feature of CAN is that as many errors as possible should be detected and handled by the CAN chip. Every error that is detected by a network node will be noticed to the rest of the network immediately. After this error message all nodes discard the received bits. The transmitter of the message is also informed about this error (by reading back the transmitted bits) and he will repeat his message transfer when the bus is free again. This will be handled automatically by the CAN controller so this is not part of the application software.

The notification of a detected error is done by an error frame that is coded by a coding violation. The bit stuffing rule is violated. The structure of an error frame can be seen in Fig. 0.4.

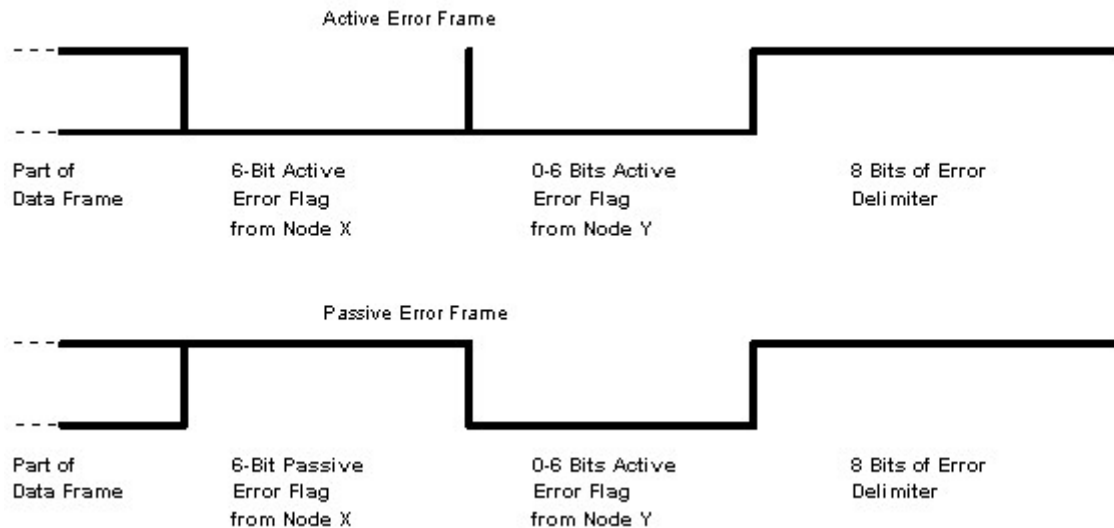


Fig. 0.4 Can Error Frames

With a sequence of 6 or more dominant bits the actual bits of the transmitted message will be overwritten. These 6 consecutive dominant bits are received by any bus node and a stuff bit violation is detected.

If there is a permanent local disturbance of a network node this would result in repeating error frames all the time. To prevent from this behavior the network nodes are disconnected from the bus activity step by step. With this algorithm it is possible that the intact network nodes are not longer disturbed by the defective node. After the first step of disconnecting the network is node is only allowed to send so called passive error frames (see Fig. 0.4).

The error process is divided three parts: Error Detection - Error Handling - Error Limitation. The error management unit is able to detect five different error types:

– Error Detection

The error management has the ability to detect five different errors

- Bit Error
A node that is sending a bit on the bus also monitors the bus. A bit error is detected at that bit time, when the bit value that is monitored differs from the bit value sent.
- Bit Stuffing Error
A stuff error is detected at the time of 6 consecutive equal bit level in a frame field that should be coded by the method of bit stuffing.
- CRC Error
The CRC sequence received is not identical to the CRC sequence calculated.
- Form Error
A form error is detected when a fixed-form bit field (CRC delimiter, ACK delimiter, EOF field) contains one or more illegal bits.
- Acknowledgment Error
An acknowledgment error is detected by a transmitter whenever it does not monitor a dominant bit during the ACK slot.

– Error Handling

After detection of one of the above described errors an error frame will be transmitted immediately. The error frame consists of 2 different fields. The first field is given by the superposition of error flags contributed from different nodes. The second field is the error delimiter. A node detecting an error condition signals this by transmission of an active error flag. The error flag's form violates the rule of bit stuffing or destroys a bit field requiring fixed form. As a consequence all other nodes also detect an error condition and likewise start transmission of an error flag. So the sequence of dominant bits, which actually can be monitored on the bus, results from a superposition of different error flags transmitted by individual nodes. The total length of this sequence varies between a minimum of 6 and a maximum of 12 bits. Passive error flags initiated by a transmitter cause errors at the receivers when they start in a frame field which is encoded by the method of bit stuffing, because they then lead to stuff errors detected by the receivers. This requires, however, that such an error flag does not start during arbitration and another node continues transmitting, or that it starts very few bits before the end of the CRC sequence and the

last bits of the CRC sequence happen to be all recessive. Passive error flags initiated by receivers are not able to prevail in any activity on the bus line. Therefore, "error passive" receivers always have to wait 6 subsequent equal bits after detecting an error condition, until they have completed their error flag.

The error delimiter consists of 8 recessive bits. After transmission of an error flag, each node sends recessive bits and monitors the bus until it detects a recessive bit. It then starts transmitting 7 more recessive bits.

The error handling happens in the following order:

1. Error detected.
2. An error frame will be transmitted.
3. The message will be discarded by every network node.
4. The error counters of every bus node are incremented.
5. The message transmission will be repeated.

– Error Limitation

To prevent from a permanent disturbed bus by error frames because of a local disturbance of one or a group of network nodes, there is implemented a special algorithm to limit the effect of this kind of error. Each CAN controller has 3 error states:

- 1. Error Active** An error active node can normally take part in bus communication and send an active error flag when an error has been detected.
- 2. Error Passive** An error passive node can normally take part in bus communication and send an passive error flag when an error has been detected. After transmission an error passive node will wait additional time before initiating a further transmission.
- 3. Bus Off** A node is the state bus off when it is switched off from bus due to a request of a fault confinement entity. In bus off state, a node can neither send nor receive any frames.

Changing the state from error active to error passive and vice versa is done automatically by the CAN controller. A node can leave the bus off state only upon a user request (hardware or software reset). The state diagram can be seen in Fig. 0.5.

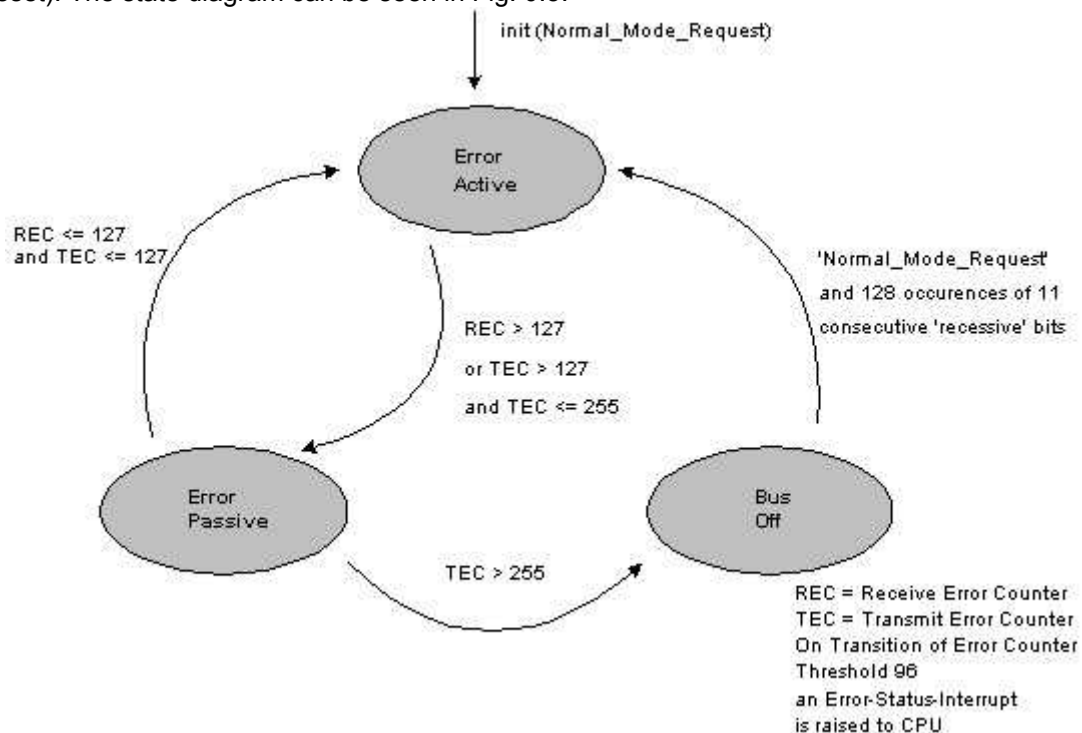


Fig. 0.5 CAN Error States

The different states depend on the values of so called error counters. There are existing one Receive Error Counter (REC) and one Transmit Error Counter (TEC) in each CAN controller. The error states are defined by the following conditions:

– Error Active

- The values of the REC and TEC are both less or equal than 127.
- Error Passive
 - At least one of the error counters REC or TEC are greater than 127 and the TEC is less or equal 255.
- Bus Off
 - The TEC is greater than 255

The error counters are modified according to the following rules (more than one rule may apply during a given frame transfer).

- a) When a transceiver detects an error, the REC will be increased by 1, except when the detected error is a bit error during the sending of an active error flag.
- b) When a receiver detects a dominant bit as the first bit after sending an error flag, the REC is increased by 8.
- c) When a transmitter sends an error flag, the TEC is increased by 8. The TEC remains unchanged, with the following exceptions:
 - Exception 1: If the transmitter is error passive and detects an acknowledgment error because of not detecting a dominant ACK slot and does not detect a dominant bit while sending its passive error flag.
 - Exception 2: If the transmitter sends an error flag because a stuff error occurred during arbitration, and should have been recessive, and was sent as recessive but monitored as dominant.
- d) If a transmitter detects a bit error while sending an active error flag or an overload flag, the TEC is increased by 8.
- e) If a receiver detects a bit error while sending an active error flag or an overload flag, the REC is increased by 8.
- f) Any node tolerates up to seven consecutive dominant bits after sending an active error flag, passive error flag or overload flag. After detecting the 14th consecutive dominant bit (in case of an active error flag or an overload flag) or after detecting the 8th consecutive dominant bit following the passive error flag, and after each additional sequence of 8 consecutive dominant bits every transmitter increases its TEC by 8 and every receiver increases its REC by 8.
- g) After a successful transmission of a frame (getting ACK and no error detected until EOF is finished) the TEC is decreased by 1 unless it was already 0.
- h) After a successful reception of a frame (reception without error up to the ACK slot and the successful sending of the ACK bit), the REC is decreased by 1, if it was between 1 and 127. If the receive error counter was 0, it remains 0; if it was greater than 127, then it will be set to a value between 119 and 127.

Under the following circumstances more than one rule will be applied:

When a network node (node A), that is receiving a message, detects an error because of a local disturbance, it will send an error flag. The rest of the network nodes detect an error at latest on the 6th bit of the error flag from node A (bit stuffing error) and will then send their error flags immediately. Node A - the locally disturbed node - detects a dominant bit after its own sent error flag and will increase the receive error counter REC by a total of 9. Increase by 1 due to rule a) and increase by 8 due to rule b). The rest of the network nodes - with no disturbance - will increase the REC only by 1, because they will not detect a dominant bit after their error flags. So the result is that node A will reach the error passive state as the first node.

To make the initialization phase of a networked system easy there are exceptions from rule c) implemented:

Under normal circumstances the preparing phases of the different nodes take different times. If during start-up only one node is on-line and if this node transmits some frame, it will get no acknowledgment, detect an error and repeat the frame. It can become error passive but not bus off. This is guaranteed through exception 1 of rule c).

The change of the error counters can be seen in Fig. 0.6.

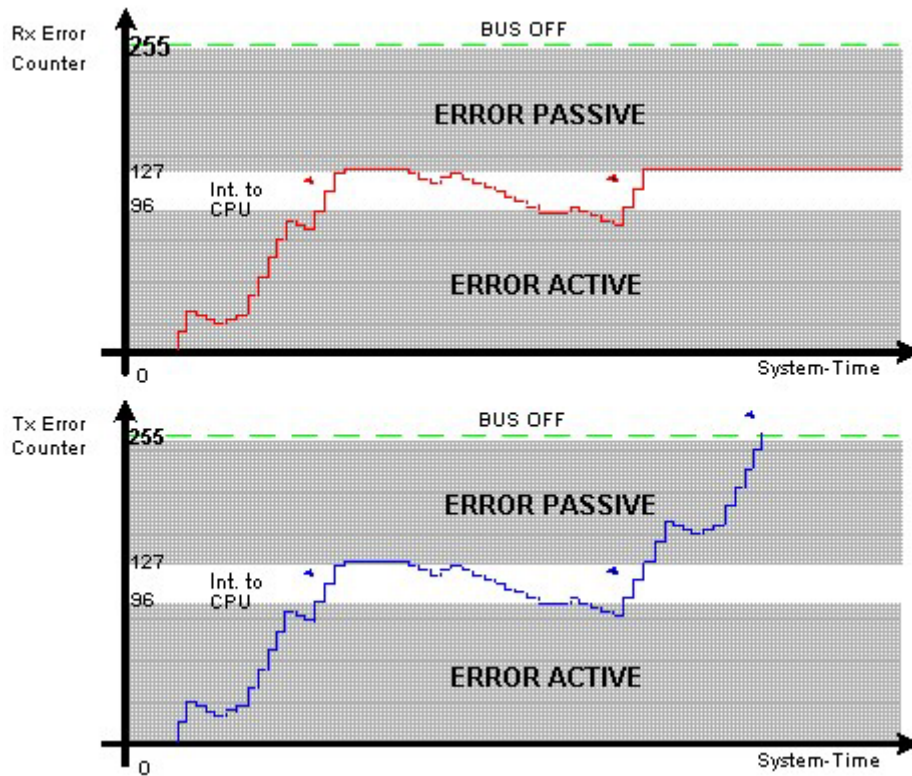


Fig. 0.6 CAN Error Counters

When one of the error counters reaches a value of 96 - just before reaching the error passive state - a special bit, the Error Status bit, will be set in the status register of the CAN controller and an interrupt - if enabled - occurs.

When the transmit error counter (TEC) exceeds the limit of 255, the Bus Status Bit will be set and again an interrupt occurs. The CAN controller is now in bus off state and will be reconnected to the bus activity only by a reset by the application.

Normally the status bits "Error Status" and "Bus Status" are the only information about errors that is shown to the application. The advantage is that the application is only loaded with error management functions if there are more errors.

Some CAN controllers provide more information about the status of the error management unit to the application, e.g. read access to the error counters or more different interrupts.

All of the above described algorithms are carried out by the CAN controller. The meaning of the error states "error passive" and "bus off" for the application depends on the type of application area. In safety critical applications the error passive state can lead to total stop of the whole system (safe state). In other applications it can be sufficient that this state is notified to the user by switching on a yellow light.

In these applications it is possible to support a so called "limp home mode", so this means the system is still running but with less performance or less functionality.

A further type of message for the exception handling is the Overload Frame. This message is used when a CAN controller, because of internal delays, is not able to process a received message. With this overload frame the transmitter is requested to repeat the message and start of the next transmission is delayed by the overload frame. The overload flag is identical to an active error frame. The only difference is that an overload flag starts at the last bit of the EOF field or in the IFS field (see Fig. 0.7).

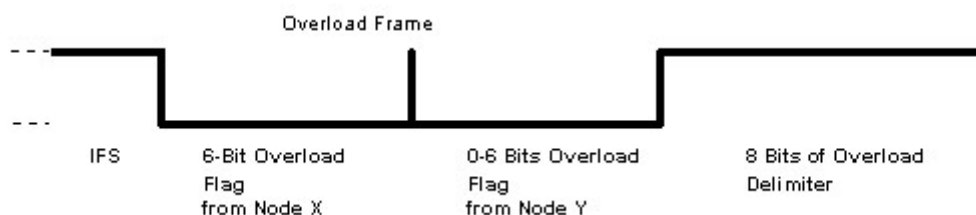


Fig. 0.7 CAN Overload Frame**Timing Considerations**

As described in chapter 0 there are 2 different frame types: Standard CAN messages and Extended CAN messages. The number of data bytes can be from 0 to 8. Due to this the data rate and the delay times depend on the type of frame and the data length.

The maximum delay time for a message can be calculated under normal circumstances only for the message with the highest priority. The delay times for the other messages can not be calculated because of the bus access mechanism of CAN (CSMA/CD+AMP). The delay times can be found out statistically by using suitable measurement equipment like simulators, emulators and analyzers. The maximum delay time for the message with the highest priority depends on the duration of the message with the maximum length and the transmission rate. In a Standard CAN network this time is calculated as follows:

- 1 Start bit
- + 11 Identifier bits
- + 1 RTR bit
- + 6 Control bits
- + 64 Data bits
- + 15 CRC bits
- + 19 (maximum) Stuff bits
- + 1 CRC delimiter
- + 1 ACK slot
- + 1 ACK delimiter
- + 7 EOF bits
- + 3 IFS (Inter Frame Space) bits
- = 130 bits

The maximum delay time for a bus access of the message with the highest priority is 130 bit times (e.g. 130 μ s with maximum transmission rate of 1 MBit/s).

In an Extended CAN network the time is calculated as follows:

- 1 Start bit
- + 11 Identifier bits
- + 1 SRR bit
- + 1 IDE bit
- + 18 Identifier bits
- + 1 RTR bit
- + 6 Control bits
- + 64 Data bits
- + 15 CRC bits
- + 23 (maximum) Stuff bits
- + 1 CRC delimiter
- + 1 ACK slot
- + 1 ACK delimiter
- + 7 EOF bits
- + 3 IFS (Inter Frame Space) bits
- = 154 bits

This results in a maximum delay time for the message with the highest priority of 154 bit times (e.g. 154 μ s with maximum transmission rate of 1 MBit/s).

The data rate is the ratio of data bits to the total number of bits of the message (including all necessary framing bits) and depends on the type of message, data length and transmission rate. The following table shows different data rates for a transmission rate of 1 MBit/s (without stuff bits):

Byte/Frame	Std.-ID	Ext.-ID	
0	0,00	0,00	Kbit/s
1	145,45	106,67	Kbit/s
2	253,97	192,77	Kbit/s
3	338,03	263,74	Kbit/s
4	405,06	323,23	Kbit/s
5	459,77	373,83	Kbit/s
6	505,26	417,39	Kbit/s
7	543,69	455,28	Kbit/s
8	576,58	488,55	Kbit/s

Oct. 2002 correction of the table: Base of the calculation now (Bit per frame) / (max.Framebits - Stuffbits) * Baudrate = row datatransfer

Example for 1 Databyte with Std.-ID at 1000 kBaud:

8 bit / (74 bit/frame - 19 bit) * 1000 kBit/s = 145,5 kBit/s

For a message with no data field a data rate can not be calculated. Such a type of message makes sense anyway: These messages can be used for so called "Life Guarding" mechanism or for synchronization of network nodes.

If we are looking at the time that is required from the microcontroller for the receive message handling, the following values have to be taken in consideration:

The worst case for e.g. the interrupt service routine for receive objects arises under the circumstances of 1 MBit/s transmissionrate, messages with no data field and a busload of 100%.

This results in Standard CAN networks in a receive interrupt every 47 µs. When the interrupt service routine takes 47 µs to be finished there is no time left for the application. In Extended CAN networks this time is 67 µs.

The tasks that have to be done depend in general on the type of CAN controller (Full CAN controller or Basic CAN controller). In an application using a Full CAN controller the message filtering will be done completely by the CAN controller. The result is, that a receive interrupt occurs only, if there was message on the bus, that is of interest for this application. The interrupt service routine has to read out the mailbox and inform the application about this event.

In applications using Basic CAN controllers, the message filtering must be done by the application.

This means the interrupt service routine has to read out the identifier and compare this result with a list of identifiers that is stored in the data area of the application. When this comparison is done then the same tasks have to be done like in Full CAN applications.

In case of errors on the bus the recovery time is in the following range:

Corrupted frames are flagged by any transmitting node and any normally operating (error active) receiving node. Such frames are aborted and will be re-transmitted according to the implemented recovery procedure (see 0). The recovery time, from detecting an error until the possible start of the next frame, is typically 17 bit times to 23 bit times (in case of a heavily disturbed bus up to 29 bit times), if there are no further errors.

Bit Timing and Synchronization

A CAN network consists of several network nodes. Each network node is clocked with its individual oscillator. Due to this phase shifts can occur in different network nodes. Each CAN controller supports a special synchronization algorithm to compensate phase shifts while receiving a CAN frame.

To ensure the proper functionality of a CAN network it is also important to sample a bit at the right position inside of a bit time.

The transmission rate is also an important parameter of a networked system. Supported transmission rates are from less than 1 kBit/s up to a maximum of 1000 kBit/s.

All these parameters can be individually programmed and are performed by the Bit Timing Logic (BTL). According to the CAN specification, a bit time is subdivided into three segments (see Fig. 0.8). The InSync segment, the Time Segment 1 (TSeg1) and Time Segment 2 (TSeg2). The segments are

composed of a certain (programmable) number of BTL cycles. The BTL cycles are defined from the quartz oscillator and a prescaler (Baud Rate Prescaler).

The sample point is located between TSeg1 and TSeg2. A programmable part of each Time Segment the Synchronization Jump Width (SJW) is used for resynchronization. With this value a bit time can be lengthened or shortened to compensate phase shifts while receiving a CAN frame.

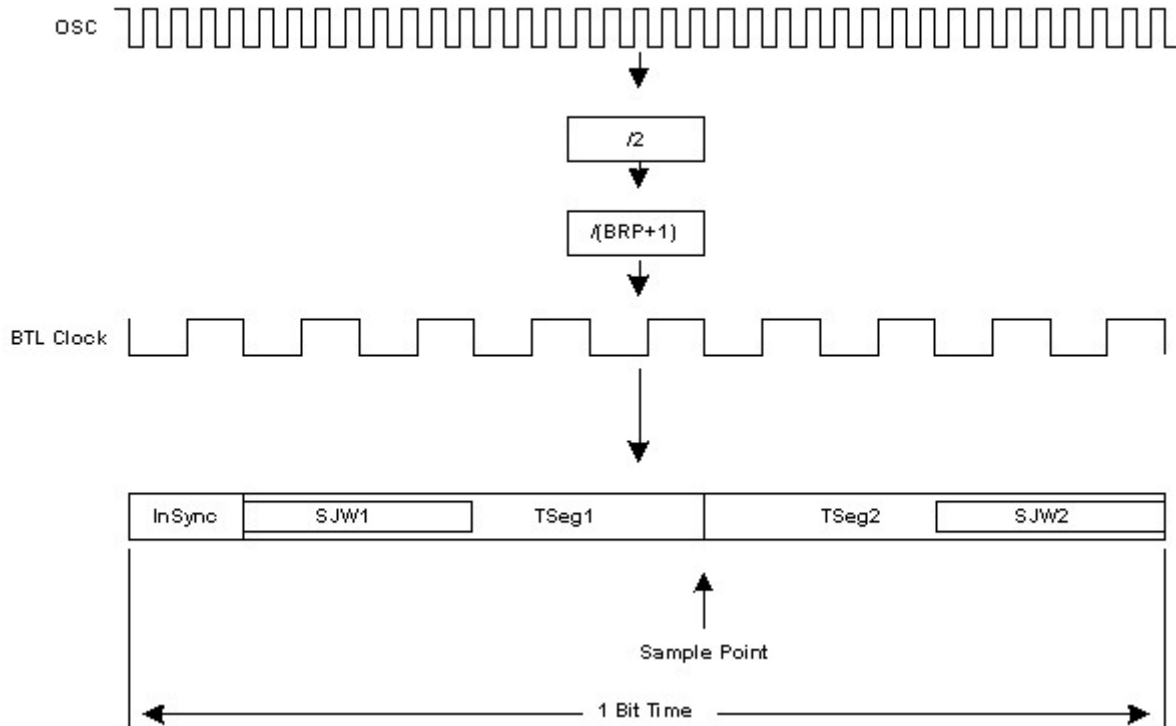


Fig. 0.8 Bit Timing

The programmable values for the segments are the following:

- InSync 1 BTL cycle
- SJW1 1 - 4 BTL cycles
- TSeg1 1 - 16 BTL cycles
- TSeg2 1 - 8 BTL cycles
- SJW2 1 - 4 BTL cycles
- (SJW1 and SJW2 have always identical values)

The bit time (this means the transmission rate) is determined by the quartz oscillator (tXOSC), the Baud Rate Prescaler (BRP) and the number of BTL cycles (sum of TSeg1 and TSeg2).

$$\begin{aligned} \text{BitTime} &= t_{\text{InSync}} + T_{\text{TSeg1}} + T_{\text{TSeg2}} \\ t_{\text{InSync}} &= 1 * t_{\text{BTL}} \\ T_{\text{TSeg1}} &= (T_{\text{SEG1}} + 1) * t_{\text{BTL}} \\ T_{\text{TSeg2}} &= (T_{\text{SEG2}} + 1) * t_{\text{BTL}} \\ t_{\text{BTL}} &= (\text{BRP} + 1) * 2 * t_{\text{XOSC}} \end{aligned}$$

TSEG1, TSEG2 and BRP are the programmed numerical values from the respective fields of the Bit Timing registers.

The transmitter will always transmit the CAN frames with the programmed values of TSeg1 and TSeg2.

Two types of synchronization are supported:

- Hard synchronization
is done with a falling edge on the bus while the bus is idle, which is interpreted as a Start of Frame (SOF). It restarts the internal Bit Time Logic.
- Soft synchronization

is used to lengthen or shorten a bit time while a CAN frame is received.

An edge is expected in the InSync cycle of a bit time. In this case a receiver starts TSeg1 and samples the bit at the end of this segment. Then he starts TSeg2. After this segment the next bit is expected. In case of a faster or slower transmitter the possible edge is located outside of the InSync cycle. In this case the resynchronization is done. The bit time of the receiver is shortened or lengthened with a maximum of the programmed value of the Synchronization Jump Width (SJW). In Fig. 0.9 the transmitter bit time is longer than the bit time of the receiver (transmitter has a slower transmission rate). In this case the edge of the next bit is detected in the SJW1 segment. The SJW1 segment will now be restarted again and the actual bit length of the receiver is increased. Lengthening the bit time is only active for exactly one bit that follows the edge. Further bits of equal polarity will be sampled with the nominal bit timing that is programmed in the receiver CAN controller. The next resynchronization is done on the next detected edge.

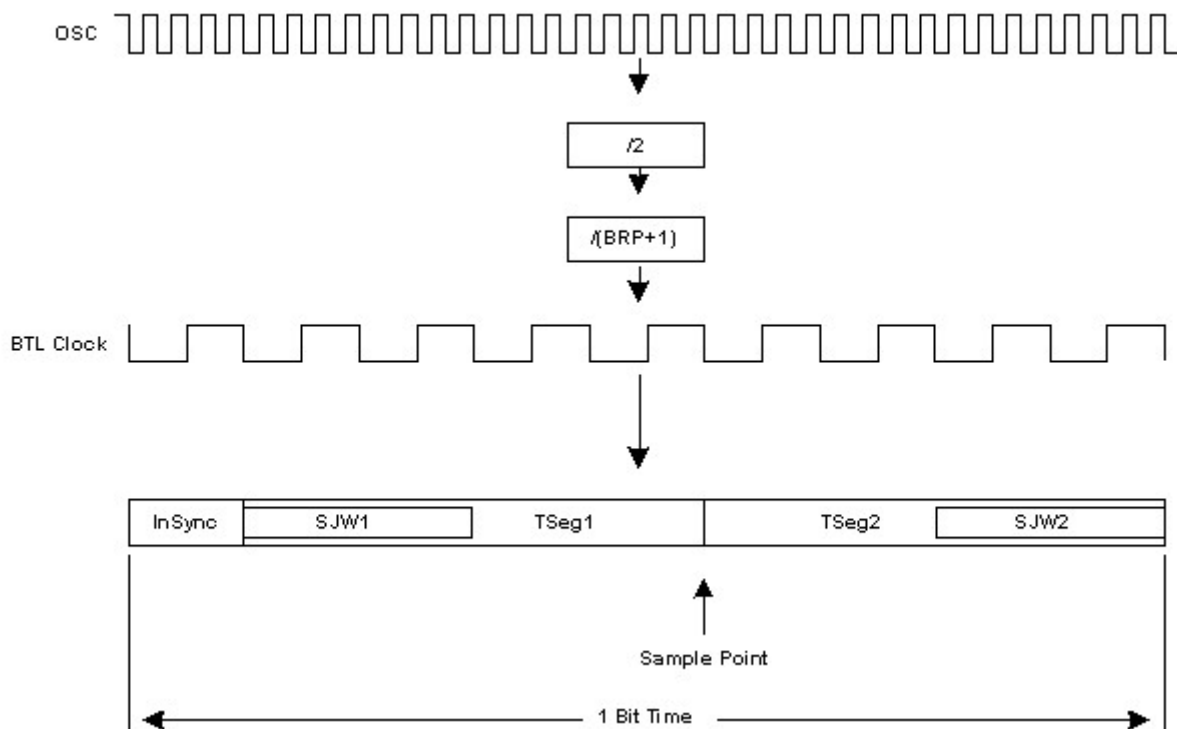


Fig. 0.9 Resynchronization to a Slower Transmitter

In Fig. 0.10 the transmitter bit time is shorter than the bit time of the receiver (transmitter has a faster transmission rate). In this case the edge of the next bit is detected in the SJW2 segment. The SJW2 segment will now be stopped and the next bit starts immediately. So the actual bit length of the receiver is decreased. Shortening the bit time is only active for exactly one bit that follows the edge. Further bits of equal polarity will be sampled with the nominal bit timing that is programmed in the receiver CAN controller. The next resynchronization is done on the next detected edge.

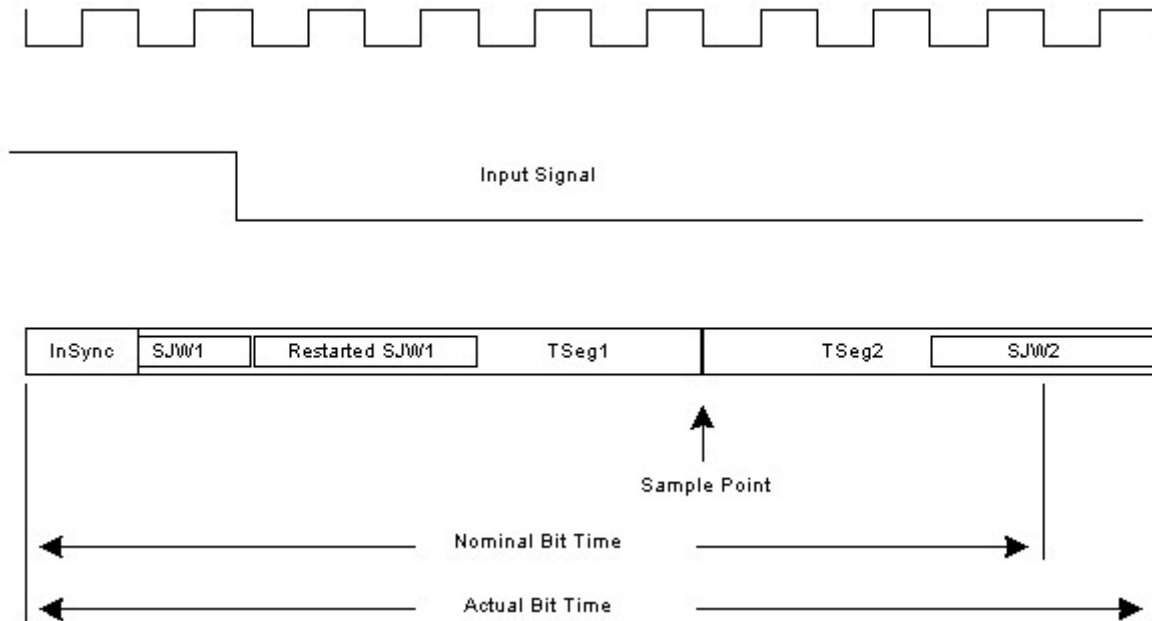


Fig. 0.10 Resynchronization to a Faster Transmitter

For higher transmission rates (> 100 kBit/s) it is recommended to use only the falling edges for resynchronization. Which edge is used (either the falling edge or both edges) can be programmed. Also a three sample mode can be used with lower baud rates. In this case the CAN controller samples a bit three times and does a majority decision. With this feature short disturbances (spikes) can be eliminated.

The minimum and maximum transmission rate that can be received (and synchronized) by a receiver depends on the ratio of the length of the synchronization segments to the length of bit time and on the possible number of bits with no edge for synchronization.

With all these features of the Bit Timing Logic it is possible to place the sample point at the right position inside of a bit depending on the quality of the bus signals. Oscillator tolerances between the different network nodes can be eliminated by using the synchronization mechanism of the CAN controller. With this the CAN controller can be optimized to work correctly in different CAN networks.

For more detailed information please contact us and questions around your design - please contact us or attend our worldwide CAN seminars and workshops.