**CAN BUS PROTOCOL**

1. CAN Protocol Basic Properties
2. Bus Termination
3. Wired AND Logic
4. Bit stuffing
5. Arbitration prosses
6. Physical and Data Link Layer
7. Standard Frame Format
8. Extended Frame Format
9. Frame Types
10. ERROR Frame
11. Overload and Interspace Frame
12. Bit/Stuff/CRC/ Form/ ACK error
13. Synchronization
14. Resynchronization
15. Bit Time and Bit Construction
16. CAN Bit segments (SYNC, PROP, PH\_SEG1, PH\_SEG2)
17. Edge phase error adjustment
18. Bit Lengthening
19. Bit Shortening
20. CAN Controllers
21. Fault confinement, TEC & REC, need of fault confinement
22. CAN Controller State (Error Active/Passive, Buss off)

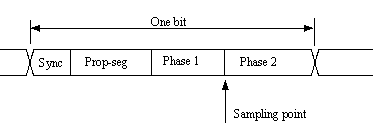
**CAN Bit Timing**

The Layout of a Bit

Each bit on the CAN bus is, for timing purposes, divided into at least 4 quanta. The quanta are logically divided into four groups or segments –

* the Synchronization Segment
* the Propagation Segment
* the Phase Segment 1
* the Phase Segment 2

Here is a picture of a CAN data bit:

[](https://www.kvaser.com/wp-content/uploads/2014/01/bit-timing-1.gif)

### Bit Timing Calculator

Use the calculator to calculate all possible sets of CAN bus parameters for a given input frequency and a given bus speed.

The Synchronization Segment, which always is one quantum long, is used for synchronization of the clocks. A bit edge is expected to take place here when the data changes on the bus.

The Propagation Segment is needed to compensate for the delay in the bus lines.

The Phase Segments may be shortened (Phase Segment 1) or lengthened (Phase Segment 2) if necessary to keep the clocks in sync.

The bus levels are sampled at the border between Phase Segment 1 and Phase Segment 2.

Most CAN controllers also provide an option to sample three times during a bit. In this case, the sampling occurs on the borders of the two quanta that precedes the sampling point, and the result is subject to majority decoding (at least this is the case for the 82527.)

Clock Synchronization

In order to adjust the on-chip bus clock, the CAN controller may shorten or prolong the length of a bit by an integral number of quanta. The maximum value of these bit time adjustments is termed the Synchronization Jump Width, SJW.

Hard synchronization occurs on the recessive-to-dominant transition of the start bit. The bit time is restarted from that edge.

Resynchronization occurs when a bit edge doesn’t occur within the Synchronization Segment in a message. One of the Phase Segments are shortened or lengthened with an amount that depends on the phase error in the signal; the maximum amount that may be used is determined by the Synchronization Jump Width parameter.

Bit Timing Register Calculation

Most CAN controllers allows the programmer to set the bit timing using the following parameters:

* A clock presale value
* The number of quanta before the sampling point
* The number of quanta after the sampling point
* The number of quanta in the *Synchronization Jump Width*, SJW

Usually two registers are provided for this purpose: btr0 and btr1. Things tend to vary slightly between different controllers, however, so read your data sheets carefully.

On the 82c200 and SJA1000, both from NXP (nee Philips), the register layout is like this:

|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| btr0 | SJW1 | SJW0 | BRP5 | BRP4 | BRP3 | BRP2 | BRP1 | BRP0 |
| btr1 | SAM | TSEG22 | TSEG21 | TSEG20 | TSEG13 | TSEG12 | TSEG11 | TSEG10 |

* BRP0.BRP5 sets the clock prescaler value
* SJW0.SJW1 sets the length of the SJW
* TSEG10.TSEG13 sets the number of quanta before the sampling point (the start bit is not included)
* TSEG20.TSEG22 sets the number of quanta after the sampling point.
* SAM is set to 1 if three samples is to be obtained and to 0 if one sample is enough.

*Note: the actual value of these parameters is one more than the value written into the register.*

Example: if the oscillator signal fed to the SJA1000 is 16 MHz, and we want a bit rate of 250 Kbit/s, with a sampling point close to 62% of the whole bit, and a SJW of 2 quanta, we can set –

BRP = 4, which gives a quantum length of 2 \* 4 / 16000000 s = 500 ns (2 is the external clock divider, see SJA1000 datasheet for more information)  
TSEG1 = 5, which gives 5 quanta before the sampling point, and  
TSEG2 = 3, which gives 3 quanta after the sampling point.

Each bit will then comprise 5 + 3 = 8 quanta, which results in the desired bit rate of 1 / (8 \* 500 ns) = 250 Kbit/s. The register values should then be

|  |  |
| --- | --- |
| btr0 = | (SJW – 1) \* 64 + (BRP -1) = (2-1) \*64 + (4-1) = 67 = 0x43 |
| btr1 = | SAM \* 128 + (TSEG2 – 1) \* 16 + (TSEG1 – 1) = 0\*128 + (3-1) \*16 + (4-1) = *(“4” because the start bit isn’t included)* 35 = 0x23 |

The sampling point is at 5/8 = 62.5% of a bit.

**CAN Bus Error Handling**

How CAN Handles Errors

Error handling is built into in the CAN protocol and is of great importance for the performance of a CAN system. The error handling aims at detecting errors in messages appearing on the CAN bus, so that the transmitter can retransmit an erroneous message. Every CAN controller along a bus will try to detect errors within a message. If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Flag (if they haven’t already detected the original error) and take appropriate action, i.e. discard the current message.

Each node maintains two error counters: The Transmit Error Counter and the Receive Error Counter. There are several rules governing how these counters are incremented and/or decremented. A transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault! When any Error Counter raises over a certain value, the node will first become “error passive”, that is, it will not actively destroy the bus traffic when it detects an error, and then “bus off”, which means that the node doesn’t participate in the bus traffic at all.

Using the error counters, a CAN node can not only detect faults but also perform error confinement.

Error Detection Mechanisms

The CAN protocol defines no less than five different ways of detecting errors. Two of these works at the bit level, and the other three at the message level.

1. Bit Monitoring.
2. Bit Stuffing.
3. Frame Check.
4. Acknowledgement Check.
5. Cyclic Redundancy Check.

Bit Monitoring

Each transmitter on the CAN bus monitors (i.e. reads back) the transmitted signal level. If the bit level read differs from the one transmitted, a Bit Error is signaled. (No bit error is raised during the arbitration process.)

Bit Stuffing

When five consecutive bits of the same level have been transmitted by a node, it will add a sixth bit of the opposite level to the outgoing bit stream. The receivers will remove this extra bit. This is done to avoid excessive DC components on the bus, but it also gives the receivers an extra opportunity to detect errors: if more than five consecutive bits of the same level occurs on the bus, a Stuff Error is signaled.

Frame check

Some parts of the CAN message have a fixed format, i.e. the standard defines exactly what levels must occur and when. (Those parts are the CRC Delimiter, ACK Delimiter, End of Frame, and also the Intermission, but there are some extra special error checking rules for that.) If a CAN controller detects an invalid value in one of these fixed fields, a Form Error is signaled.

Acknowledgement Check

All nodes on the bus that correctly receives a message (regardless of their being “interested” of its contents or not) are expected to send a dominant level in the so-called Acknowledgement Slot in the message. The transmitter will transmit a recessive level here. If the transmitter can’t detect a dominant level in the ACK slot, an Acknowledgement Error is signaled.

Cyclic Redundancy Check

Each message features a 15-bit Cyclic Redundancy Checksum (CRC), and any node that detects a different CRC in the message than what it has calculated itself will signal an *CRC Error*.

Error Confinement Mechanisms

Every CAN controller along a bus will try to detect the errors outlined above within each message. If an error is found, the discovering node will transmit an Error Flag, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Flag (if they haven’t already detected the original error) and take appropriate action, i.e. discard the current message.

Each node maintains two error counters: The Transmit Error Counter and the Receive Error Counter. There are several rules governing how these counters are incremented and/or decremented. A transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault!

A node starts out in Error Active mode. When any one of the two Error Counters raises above 127, the node will enter a state known as Error Passive and when the Transmit Error Counter raises above 255, the node will enter the Bus Off state.

* An Error Active node will transmit Active Error Flags when it detects errors.
* An Error Passive node will transmit Passive Error Flags when it detects errors.
* A node which is Bus Off will not transmit anything on the bus at all.

The rules for increasing and decreasing the error counters are somewhat complex, but the principle is simple: transmit errors give 8 error points, and receive errors give 1 error point. Correctly transmitted and/or received messages causes the counter(s) to decrease.  
Example (slightly simplified): Let’s assume that node A on a bus has a bad day. Whenever A tries to transmit a message, it fails (for whatever reason). Each time this happens, it increases its Transmit Error Counter by 8 and transmits an Active Error Flag. Then it will attempt to retransmit the message... and the same thing happens.

When the Transmit Error, Counter raises above 127 (i.e. after 16 attempts), node A goes Error Passive. The difference is that it will now transmit Passive Error Flags on the bus. A Passive Error Flag comprises 6 recessive bits and will not destroy other bus traffic – so the other nodes will not hear A complaining about bus errors. However, A continues to increase its Transmit Error Counter. When it raises above 255, node A finally gives in and goes Bus Off.

What does the other nodes think about node A? – For every active error flag that A transmitted, the other nodes will increase their Receive Error Counters by 1. By the time that A goes Bus Off, the other nodes will have a count in their Receive Error Counters that is well below the limit for Error Passive, i.e. 127. This count will decrease by one for every correctly received message. However, node A will stay bus off.

Most CAN controllers will provide status bits (and corresponding interrupts) for two states:

* “Error Warning” – one or both error counters are above 96
* Bus Off, as described above.

Some – but not all! – controllers also provide a bit for the Error Passive state. A few controllers also provide direct access to the error counters.

The CAN controller’s habit of automatically retransmitting messages when errors have occurred can be annoying at times. There is at least one controller on the market (the SJA1000 from Philips) that allows for full manual control of the error handling.

Bus Failure Modes

The ISO 11898 standard enumerates several failure modes of the CAN bus cable:

1. CAN\_H interrupted
2. CAN\_L interrupted
3. CAN\_H shorted to battery voltage
4. CAN\_L shorted to ground
5. CAN\_H shorted to ground
6. CAN\_L shorted to battery voltage
7. CAN\_L shorted to CAN\_H wire
8. CAN\_H and CAN\_L interrupted at the same location
9. Loss of connection to termination network

For failures 1-6 and 9, it is “recommended” that the bus survives with a reduced S/N ratio, and in case of failure 8, that the resulting subsystem survives. For failure 7, it is “optional” to survive with a reduced S/N ratio.

In practice, a CAN system using 82C250-type transceivers will not survive failures 1-7 and may or may not survive failures 8-9.

There are “fault-tolerant” drivers, like the TJA1053, that can handle all failures though. Normally you pay for this fault tolerance with a restricted maximum speed; for the TJA1053 it is 125 Kbit/s.

Controller Area Network

# CAN Data Link Layer

• Bus Access Arbitration

• Frame Formats

• Error Detection

• Error Handling

• Protocol Versions

(Standard /Extended

# CAN Physical Layer

• Message Coding

• Synchronization

• Bit Timing / Bit Construction

• CAN Bus Lines

# CAN User Benefits

CAN is low cost

• Fast serial bus with two wires: good price/performance ratio

• Low cost protocol devices (controllers, transceivers) available mainly driven

by high volume production in the automotive market

CAN is reliable

• Sophisticated error detection and error handling mechanisms results in high

reliability transmission

• Example: 500 kbit/s, 25% bus load, 2000 hours per year:

One undetected error every 1000 years

• Erroneous messages are detected and repeated

• System-wide data consistency (every bus node is informed about an error)

• Faulty nodes automatically withdraw from bus communication

• High immunity to Electromagnetic Interference

CAN means real-time

• maximum data rate is 1 MBit/s @ 40m bus length

(still about 40 kBit/s @ 1000m bus length)

• Short message length (0 to 8 data bytes / message)

(Larger data can be split up into several messages)

• Low latency between transmission request and actual start of transmission

• Bus access handled via CSMA/CD w/ AMP method (message with the

highest priority wins arbitration without losing any time)

q CAN is flexible

• CAN allows Multi-Master Operation

(every CAN node is able to access the bus individually)

• CAN Nodes can easily be connected / disconnected

(i.e. plug & play)

• Number of nodes not limited by the protocol

CAN means Multicast / Broadcast Capability

• CAN is not node-oriented but message-oriented

• Message identifier specifies contents & priority of the message

• Messages can be easily sent to multiple / all nodes simultaneously

• All nodes simultaneously receive and work on common data

q CAN is standardized

• ISO-11898 (high speed applications)

• ISO-11519-2 (low speed applications)

# How it all began...

•The development of CAN began when more and more electronic devices were implemented into modern motor vehicles. Examples of such devices include engine management systems, active suspension, ABS, gear control, lighting control, air conditioning, airbags and central locking. All this means more safety and more comfort for the driver and of course a reduction of fuel consumption and exhaust emissions.

•To improve the behavior of the vehicle even further, it was necessary for the different control systems (and their sensors) to exchange information. This was usually done by discrete interconnection of the different systems (i.e. point to point wiring). The requirement for information exchange has then grown to such an extent that a cable network with a length of up to several miles and many connectors was required. This produced growing problems concerning material cost, production time and reliability.

•The solution to this problem was the connection of the control systems via a serial bus system. This bus had to fulfill some special requirements due to its usage in a vehicle.

•With the use of CAN, point-to-point wiring is replaced by one serial bus connecting all control systems. This is accomplished by adding some CAN-specific hardware to each control unit that provides the "rules" or the protocol for transmitting and receiving information via the bus.

# Overview

•CAN or Controller Area Network is an advanced serial bus system that efficiently supports distributed control systems. It was initially developed for the use in motor vehicles by Robert Bosch GmbH, Germany, in the late 1980s, also holding the CAN license.

•CAN is internationally standardized by the International Standardization Organization (ISO) and the Society of Automotive Engineers (SAE).

•The CAN protocol uses the Data Link Layer and the Physical Layer in the ISO - OSI model. There are also a number of higher level protocols available for CAN.

•CAN is most widely used in the automotive and industrial market segments. Typical applications for CAN are motor vehicles, utility vehicles, and industrial automation. Other applications for CAN are trains, medical equipment, building automation, household appliances, and office automation. Due to the high-volume production in the automotive and industrial markets, low cost protocol devices are available.

•There are about 100 million CAN nodes in use worldwide.

•Examples of vehicle bus systems, other than CAN, are A-BUS from Volkswagen, VAN or Vehicle Area Network, from Peugeot and Renault, and J1850 from Chrysler, General Motors and Ford.

•CAN is clearly the leading vehicle bus protocol in Europe.

# Basic Concepts

•CAN is a multi-master bus with an open, linear structure with one logic bus line and equal nodes. The number of nodes is not limited by the protocol.

•In the CAN protocol, the bus nodes do not have a specific address. Instead, the address information is contained in the identifiers of the transmitted messages, indicating the message content and the priority of the message.

•The number of nodes may be changed dynamically without disturbing the communication of the other nodes.

•Multicasting and Broadcasting is supported by CAN.

-

•CAN provides sophisticated error-detection and error handling mechanisms such as CRC check, and high immunity against electromagnetic interference. Erroneous messages are automatically retransmitted. Temporary errors are recovered. Permanent errors are followed by automatic switch-off of defective nodes. There is guaranteed system-wide data consistency.

•The CAN protocol uses Non-Return-to-Zero or NRZ bit coding. For synchronization purposes, Bit Stuffing is used.

•There is a high data transfer rate of 1000 kilobit per second at a maximum bus length of 40 meters or 130 feet when using a twisted wire pair which is the most common bus medium used for CAN. Message length is short with a maximum of 8 data bytes per message and there is a low latency between transmission request and start of transmission.

•The bus access is handled via the advanced serial communications protocol Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration. This means that collision of messages is avoided by bitwise arbitration without loss of time.

# Basic Concepts -

# CAN Bus Characteristics - Wired-AND

•There are two bus states, called "dominant" and "recessive".

•The bus logic uses a "Wired-AND" mechanism, that is, "dominant bits" (equivalent to the logic level "Zero") overwrite the "recessive" bits (equivalent to the logic level "One”).

•Only if all nodes transmit recessive bits (ones), the Bus is in the recessive state.

•As soon as one node transmits a dominant bit (zero), the bus is in the dominant state.

# Bus Access and Arbitration: CSMA/CD w/ AMP

•The CAN protocol handles bus accesses according to the concept called “Carrier Sense Multiple Access with Arbitration on Message Priority”. This arbitration concept avoids collisions of messages whose transmission was started by more than one node simultaneously and makes sure the most important message is sent first without time loss. In the picture above, you see the trace of the transmit pins of three bus nodes called A, B and C, and the resulting bus state according to the wired-AND principle.

•If two or more bus nodes start their transmission at the same time after having found the bus to be idle, collision of the messages is avoided by bitwise arbitration. Each node sends the bits of its message identifier and monitors the bus level.

•At a certain time, nodes, A and C send a dominant identifier bit. Node B sends a recessive identifier bit but reads back a dominant one. Node B loses bus arbitration and switches to receive mode. Some bits later node C loses arbitration against node A. This means that the message identifier of node A has a lower binary value and therefore a higher priority than the messages of nodes B and C. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message.

•Nodes B and C automatically try to repeat their transmission once the bus returns to the idle state. Node B loses against node C, so the message of node C is transmitted next, followed by node B’s message.

•It is not permitted for different nodes to send messages with the same identifier as arbitration could fail leading to collisions and errors.

# Frame Formats - Overview

## Frame Formats - Data Frame

•A "Data Frame" is generated by a CAN node when the node wishes to transmit data. The Standard CAN Data Frame is shown above. The frame begins with a dominant Start of Frame bit for hard synchronization of all nodes.

•The Start of Frame bit is followed by the Arbitration Field consisting of 12 bits: The 11-bit Identifier, which reflects the contents and priority of the message, and the Remote Transmission Request bit. The Remote transmission request bit is used to distinguish a Data Frame (RTR = dominant) from a Remote Frame (RTR = recessive).

•The next field is the Control Field, consisting of 6 bits. The first bit of this field is called the IDE bit (Identifier Extension) and is at dominant state to specify that the frame is a Standard Frame. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) and specify the number of bytes of data contained in the message (0 - 8 bytes).

•The data being sent follows in the Data Field which is of the length defined by the DLC above (0, 8, 16, ...., 56 or 64 bits).

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

•The next field is the Acknowledge Field. During the ACK Slot bit the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not). From this CAN belongs to the "in-bit-response" group of protocols. The recessive Acknowledge Delimiter completes the Acknowledge Slot and may not be overwritten by a dominant bit.

•Seven recessive bits (End of Frame) end the Data Frame.

## Frame Formats - Remote Frame

•Generally, data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node to request the data from the source by sending a Remote Frame.

•There are 2 differences between a Data Frame and a Remote Frame. Firstly, the RTR-bit is transmitted as a dominant bit in the Data Frame and secondly in the Remote Frame there is no Data Field. In the very unlikely 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.

•If a node wishes to request the data from the source, it 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 as a response to this remote request.

## Frame Formats - Error Frame

•An Error Frame is generated by any node that detects a bus error. The Error Frame consists of 2 fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter consists of 8 recessive bits and allows the bus nodes to restart bus communications cleanly after an error. There are, however, two forms of Error Flag fields. The form of the Error Flag field depends on the “error status” of the node that detects the error.

•If an “error-active” node detects a bus error then the node interrupts transmission of the current message by generating an “active error flag”. The “active error flag” is composed of six consecutive dominant bits. This bit sequence actively violates the bit stuffing rule. All other stations recognize the resulting bit stuffing error and in turn generate Error Frames themselves. The Error Flag field therefore consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame bus activity returns to normal and the interrupted node attempts to resend the aborted message.

•If an “error passive” node detects a bus error then the node transmits an “passive Error Flag” followed, again, by the Error Delimiter field. The “passive Error Flag” consists of six consecutive recessive bits, and therefore the Error Frame (for an “error passive” node) consists of 14 recessive bits (i.e. no dominant bits). From this it follows that, unless the bus error is detected by the node that is transmitting (i.e. is the bus master), the transmission of an Error Frame by an “error passive” node will not affect any other node on the network. If the bus master node generates an “error passive flag” then this may cause other nodes to generate error frames due to the resulting bit stuffing violation.

## Frame Formats - Overload Frame

•An Overload Frame has the same format as an “active” Error Frame. An Overload Frame, however can only be generated during Interframe Space. This is the way then 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 2 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 (as for “active error flag”, again giving a maximum of twelve dominant bits). The Overload Delimiter consists of eight recessive bits. An Overload Frame can be generated by a node if due to internal conditions the node is not yet able to start reception of the next message. A node may generate a maximum of 2 sequential Overload Frames to delay the start of the next message.

## Frame Formats - Interframe Space

•Interframe Space separates a preceding frame (of whatever type) from a following Data or Remote Frame. Interframe space is composed of at least 3 recessive bits, these bits are termed the Intermission. This time is provided to allow nodes time for internal processing before the start of the next message frame. After the Intermission, for error active CAN nodes the bus line remains in the recessive state (Bus Idle) until the next transmission starts.

•The Interframe Space has a slightly different format for error passive CAN nodes which were the transmitter of the previous message. In this case, these nodes have to wait another eight recessive bits called Suspend Transmission before the bus turns into bus idle for them after Intermission and they are allowed to send again. Due to this mechanism error active nodes have the chance to transmit their messages before the error passive nodes are allowed to start a transmission.

# Error Detection - Overview

## Error Detection - Cyclic Redundancy Check

•With the Cyclic Redundancy Check, the transmitter calculates a check sum for the bit sequence from the start of frame bit until the end of the Data Field.

•This CRC sequence is transmitted in the CRC Field of the CAN frame.

•The receiving node also calculates the CRC sequence using the same formula and performs a comparison to the received sequence.

•If node B detects a mismatch between the calculated and the received CRC sequence, then a CRC error has occurred.

•Node B discards the message and transmits an Error Frame to request retransmission of the garbled frame.

## Error Detection - Acknowledge

•With the Acknowledge Check the transmitter checks in the Acknowledge Field of a message to determine if the Acknowledge Slot, which is sent out as a recessive bit, contains a dominant bit.

•If this is the case, at least one other node, (here node B) has received the frame correctly.

•If not, an Acknowledge Error has occurred, and the message has to be repeated. No Error Frame is generated, though.

## Error Detection - Frame Check

•Another error detection mechanism is the Frame Check. If a transmitter detects a dominant bit in one of the four segments:

CRC Delimiter,

Acknowledge Delimiter,

End of Frame or

Interframe Space

then a Form Error has occurred, and an Error Frame is generated. The message will then be repeated.

## Error Detection - Bit Monitoring

•All nodes perform Bit Monitoring: A Bit Error occurs if a transmitter

sends a dominant bit but detects a recessive bit on the bus line or,

sends a recessive bit but detects a dominant bit on the bus line.

•An Error Frame is generated, and the message is repeated.

•When a dominant bit is detected instead of a recessive bit, no error occurs during the Arbitration Field or the Acknowledge Slot because these fields must be able to be overwritten by a dominant bit in order to achieve arbitration and acknowledge functionality.

## Error Detection - Bit Stuffing Check

•If six consecutive bits with the same polarity are detected between Start of Frame and the CRC Delimiter, the bit stuffing rule has been violated.

•A stuff error occurs, and an Error Frame is generated. The message is then repeated.

# Error Handling

•Detected errors are made public to all other nodes via Error Frames.

•The transmission of the erroneous message is aborted, and the frame is repeated as soon as possible.

•Each CAN node is in one of three error states "error active", "error passive" or "bus off" according to the value of their internal error counters.

•The error-active state is the usual state after reset. The bus node can then receive and transmit messages and transmit active Error Frames (made of dominant bits) without any restrictions. During CAN communication, the error counters are updated according to quite complex rules. For each error on reception or transmission, the error counters are incremented by a certain value. For each successful transaction, the error counters are decremented by a certain value. The error active state is valid as long as both error counters are smaller than or equal to 127.

•If either the receive or the transmit error counter has reached the value of 128, the node switches to the error-passive state. In the error-passive state, messages can still be received and transmitted, although, after transmission of a message the node must suspend transmission. It must wait 8-bit times longer than error-active nodes before it may transmit another message. In terms of error signaling, only passive Error Frames (made of recessive bits) may be transmitted by an error-passive node.

•If both error counters go below 128 again due to successful bus communication, the node switches back to the error-active state.

•One feature of the CAN protocol is that faulty nodes withdraw from the bus automatically. The bus-off state is entered if the transmit error counter exceeds the value of 255. All bus activities are stopped which makes it temporarily impossible for the station to participate in the bus communication. During this state, messages can be neither received nor transmitted. To return to the error active state and to reset the error counter values, the CAN node has to be reinitialized.

## Undetected Errors - an example

•To understand the error detection capabilities of CAN, imagine a vehicle equipped with CAN running 2000 hours per year at a CAN bus speed of 500 kbps with 25% bus load.

•This will result in 1 undetected error every 1000 years.

CAN Protocol Versions

•The original CAN specifications (Versions 1.0, 1.2 and 2.0A) specify an 11-bit message identifier. This is known as "Standard CAN".

•Those Data Frames and Remote Frames, which contain an 11-bit identifier are therefore called Standard Frames.

•With these frames, 211 (=2048) different messages can be identified (identifiers 0-2047).

•However, the 16 messages with the lowest priority (2032-2047) are reserved.

•Specification V2.0A has since been updated (to version 2.0B) to remove this possible message number limitation and meet the SAE J1939 standard for the use of CAN in trucks.

•Version 2.0B CAN is referred to as "Extended CAN".

•Extended Frames, according to CAN specification V2.0-part B, contain a 29-bit identifier which allows 229 (over 536 Million) message identifiers.

•The 29-bit identifier is made up of the 11-bit identifier ("Base ID") and the 18-bit Extended Identifier ("ID Extension").

•CAN specification Version 2.0B still allows message identifier lengths of 11 bits to be used.

•There are three different types of CAN modules available.

2.0A - Considers 29-bit ID as an error

2.0B Passive - Ignores 29-bit ID messages

2.0B Active - Handles both 11 and 29-bit ID Messages

•CAN modules specified after CAN V2.0-part A are only able to transmit and receive Standard Frames according to the Standard CAN protocol.

•Messages using the 29-bit identifier sent to a Standard CAN module cause errors.

•If a device is specified after CAN V2.0-part B, there's one more distinction. Modules named "V2.0B Passive" can only transmit and receive Standard Frames but tolerate Extended Frames without generating Error Frames.

•"V2.0B Active" devices are able to transmit and receive both Standard and Extended Frames.

•Siemens offers V2.0B Active and V2.0B Passive devices.

# Message Coding

•The CAN protocol uses Non-Return-to-Zero or NRZ bit coding. This means that the signal is constant for one whole bit time and only one-time segment is needed to represent one bit.

•Usually, but not always, a "zero" corresponds to a dominant bit, placing the bus in the dominant state, and a "one" corresponds to a recessive bit, placing the bus in the recessive state.

# Bit Stuffing

•One characteristic of Non-Return-to-Zero code is that the signal provides no edges that can be used for resynchronization when transmitting a large number of consecutive bits with the same polarity.

•Therefore, Bit stuffing is used to ensure synchronization of all bus nodes.

•This means that during the transmission of a message, a maximum of five consecutive bits may have the same polarity.

•Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit of the opposite polarity into the bit stream before transmitting further bits.

•The receiver also checks the number of bits with the same polarity and removes the stuff bits again from the bit stream. This is called "destuffing".

Bit Stuffing

One characteristic of Non-Return-to-Zero code is that the signal provides no edges that can be used for resynchronization when transmitting a large number of consecutive bits with the same polarity.

•Therefore, Bit stuffing is used to ensure synchronization of all bus nodes.

•This means that during the transmission of a message, a maximum of five consecutive bits may have the same polarity.

•Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit of the opposite polarity into the bit stream before transmitting further bits.

•The receiver also checks the number of bits with the same polarity and removes the stuff bits again from the bit stream. This is called "destuffing".

# Bus Synchronization

•In contrast to many other field buses, CAN handles message transfers synchronously.

•All nodes are synchronized at the beginning of each message with the first falling edge of a frame which belongs to the Start of Frame bit.

•This is called Hard Synchronization.

•To ensure correct sampling up to the last bit, the CAN nodes need to re-synchronize throughout the entire frame. This is done on each recessive to dominant edge.

# Bit Construction

•One CAN bit time (or one high or low pulse of the NRZ code) is specified as four non-overlapping time segments.

•Each segment is constructed from an integer multiple of the Time Quantum.

•The Time Quantum or TQ is the smallest discrete timing resolution used by a CAN node.

•Its length is generated by a programmable divide of the CAN node's oscillator frequency.

•There is a minimum of 8 and a maximum of 25 Time Quanta per bit.

•The bit time, and therefore the bit rate, is selected by programming the width of the Time Quantum and the number of Time Quanta in the various segments.

# Synchronization Segment

•The first segment within a CAN bit is called the Synchronization Segment and is used to synchronize the various bus nodes.

•On transmission, at the start of this segment, the current bit level is output.

•If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment by the receiving nodes.

•The length of this segment is always 1 Time Quantum.

# Propagation Segment

•The Propagation Time Segment is used to compensate for signal delays across the network.

•This is necessary to compensate for signal propagation delays on the bus line and through the electronic interface circuits of the bus nodes.

•This segment may be 1 to 8 Time Quanta long.

# Phase Buffer Segment 1

•Phase Buffer Segment 1 is used to compensate for edge phase errors. This segment may be between 1 to 8 Time Quanta long and may be lengthened during resynchronization.

•The sample point is the point of time at which the bus level is read and interpreted as the value of the respective bit. Its location is at the end of Phase Buffer Segment 1 (between the two-Phase Buffer Segments).

# Phase Buffer Segment 2

•Phase Buffer Segment 2 is also used to compensate for edge phase errors. This segment may be shortened during resynchronization.

•Phase Buffer Segment 2 may be between 1 to 8 Time Quanta long, but the length has to be at least as long as the information processing time (see below) and may not be more than the length of Phase Buffer Segment 1.

•The information processing time begins with the sample point and is reserved for calculation of the subsequent bit level. It is less than or equal to two Time Quanta long.

# Bit Lengthening

•Because of resynchronization, Phase Buffer Segment 1 may be lengthened, or Phase Buffer Segment 2 may be shortened to compensate for oscillator tolerances within the different CAN nodes.

•If, for example, the transmitter oscillator is slower than the receiver oscillator, the next falling edge used for resynchronization may be delayed. So, Phase Buffer Segment 1 is lengthened...

in order to adjust the sample point and the end of the bit time.

# Bit Shortening

•If, on the other hand, the transmitter oscillator is faster than the receiver oscillator, the next falling edge used for resynchronization may be too early. So, Phase Buffer Segment 2 in bit N is shortened...

in order to adjust the sample point for bit N+1 and the end of the bit time

# Synchronization Jump Width

•The limit to the amount of lengthening or shortening of the phase buffer segments is set by the Resynchronization Jump Width.

•The Resynchronization Jump Width may be between 1 and 4 Time Quanta, but it may not be longer than Phase Buffer Segment 2.

# Bit Timing

•For many CAN module implementations, the Propagation Time Segment and Phase Buffer Segment 1 are combined, for ease of programming, into one segment often called Timing Segment 1.

•Phase Buffer Segment 2 is then known as Timing Segment 2.

# Why Program the Sample Position?

Programming of the sample point allows "tuning" of the characteristics to suit the bus.

•Early sampling allows more Time Quanta in the Phase Buffer Segment 2, so the Synchronization Jump Width can be programmed to its maximum of 4 Time Quanta.

•This maximum capacity to shorten or lengthen the bit time decreases the sensitivity to node oscillator tolerances, so that lower cost oscillators such as ceramic resonators may be used.

•Late sampling allows more Time Quanta in the Propagation Time Segment which allows a poorer bus topology and maximum bus length.

# Relation between Baud Rate and Bus Length

•The maximum CAN bus speed is 1 Maud, which can be achieved with a bus length of up to 40 meters when using a twisted wire pair.

•For bus lengths longer than 40 meters the bus speed must be reduced.

•A 1000-meter bus can still be realized with a 50 Baud bus speed.

•For a bus length above 1000 meters special drivers should be used.

# CAN Bus Line Characteristics - Wired-AND?

CAN is serial bus system with one logical bus line. It has an open, linear bus structure with equal bus nodes. The number of nodes on the bus is not restricted by the protocol and may be changed dynamically without disturbing the communication of other nodes. This allows easy connection and disconnection of bus nodes, e.g. for addition of system function, error recovery or bus monitoring.

•The CAN bus line has two logic states: a “recessive” state and a “dominant” state. The actual bus state is “wire-AND” of all node states. This means, that recessive bits (mostly, but not necessarily equivalent to the logic level "1") are overwritten by dominant bits (mostly logic level "0"). If no bus node is sending a dominant bit, the bus line is in the recessive state, but a dominant bit from any bus node generates the dominant bus state.

# ISO Physical Layer

•Therefore, for the CAN bus line, a medium must be chosen that is able to transmit the two possible bit states “dominant” and “recessive”. One of the most common and cheapest ways is to use a twisted wire pair. The bus lines are then called "CAN\_H" and "CAN\_L". The two bus lines CAN\_H and CAN\_L are driven by the nodes with a differential signal. The twisted wire pair is terminated by terminating resistors at each end of the bus line, typically 120 Ohms.

•But also, an optical medium would be possible for CAN. In this case, the recessive state would be represented by the signal “light off”, the dominant state by the signal “light on”.

# CAN and EMI

Due to the differential nature of transmission CAN is insensitive to electromagnetic interference, because both bus lines are affected in the same way which leaves the differential signal unaffected.

•To reduce the sensitivity against electromagnetic interference even more, the bus lines can additionally be shielded. This also reduces the electromagnetic emission of the bus itself, especially at high baud rates.

# Standardization Issues

•Vehicle bus system applications can be separated in three different categories according to their real-time capabilities.

•Class A for a low speed bus with bit rates up to 10 kbps, e.g. for body control applications,

•Class B for a low speed bus with bit rates from 10 kbps to 125 kbps, e.g. for dashboard and diagnostics,

•Class C for a high-speed bus with bit rates from 125 kbps to 1 MPs for real time applications like engine management, Gearbox, ABS etc.

•For the use of CAN in vehicles two standards have been defined for the bus interface:

•CAN High Speed according to ISO-IS 11898 for bit rates between 125 kbps and 1 MPs

•CAN Low Speed according to ISO-IS 11519-2 for bit rates up to 125 kbps

# Physical Layer according to ISO-IS 11898

This is the structure of a Controller Area Network according to ISO-IS 11898. The bus lines may be up to 40 m (130 ft) long at the maximum speed of 1 M baud and are terminated by termination resistors of 120 Ohms. The bus lines may be longer when decreasing the baud rate. Up to 30 nodes can be connected with CAN drivers according to this standard. For the connection of more nodes, stronger drivers or repeaters have to be used. To avoid reflexions the connection from the bus lines to the nodes should not exceed 0.3 m (1 ft) at 1 Mbps.

# Bus Levels according to ISO-IS 11898

•These are the bus levels according to ISO-IS 11898. A recessive bit is represented by both CAN bus lines driven to a level of about 2.5 V so that the differential voltage between CAN\_H and CAN\_L is around 0 V.

•A dominant bit is represented by CAN\_H going to about 3.5 V and CAN\_L going to about 1.5 V. This results in a differential voltage for a dominant bit of about 2V.

# CAN bus connectors according to CiA-DS 102-1

•To be able to use CAN as an industrial field bus in an open system the CAN in Automation user’s group CiA created a standard called CiA DS 102-1 which is based on the 11898-standard. One important issue in this standard is the proposal to use a 9-pole SUB-D connector for the connection of nodes to the CAN bus lines.

•The bus signals CAN\_H and CAN\_L are available on pins 7 and 2. The other pins serve as power or ground wires or are reserved for future extensions of the standard.

# Typical CAN Implementations

A typical CAN node used to control a certain application consists of different devices.

•The application itself is controlled by a microcontroller, e.g. the Siemens SAB 80C166.

•To be able to participate in the CAN communication, the microcontroller must be connected to a CAN protocol controller, e.g. the Siemens stand-alone Full-CAN controller 81C90/91.

•To meet the requirements of e.g. the ISO 11898 CAN standard, a CAN transceiver chip is used to connect the node to the CAN bus lines.

•A more sophisticated way is to use a microcontroller which already has a CAN protocol controller on-chip, e.g. one of the Siemens 8051-compatible 8-bit microcontrollers from the C500 family like the C505C or the C515C. For applications which need higher performance, one of the 16-bit C166-family members with integrated CAN module could be used, e.g. the C164CI or the C167CR. This saves costs as the printed circuit board space is used more efficiently and the user does not have to worry about setting up the communication between the microcontroller and the CAN controller.

# Basic CAN controller

•There is one more CAN characteristic concerning the interface between the CAN protocol controller and the host CPU, dividing CAN chips into "Basic-CAN" and "Full-CAN" devices. This has nothing to do with the used protocol Version though, which makes it possible to use both Basic-CAN and Full-CAN devices in the same network.

•In the Basic-CAN devices, only basic functions concerning the filtering and management of CAN messages are implemented in hardware. A Basic-CAN controller typically provides one transmit buffer for outgoing messages and one or two receive buffers for incoming messages. In the receive path, an acceptance filtering is available which allows that only certain CAN identifiers are stored in the receive buffer.

•Because there are only two buffers for the reception of messages the host controller is quite busy reading and storing the incoming messages before they get overwritten by the following ones which results in a quite high CPU load. Also, the answering of Remote Frames with the corresponding Data Frame has to be handled by the host controller. Therefore Basic-CAN devices should only be used at low baud rates and low bus loads with only a few different messages.

# Full CAN controller

•Full-CAN devices provide the whole hardware for convenient acceptance filtering and message management. For each message to be transmitted or received these devices contain one so called message object in which all information regarding the message (e.g. identifier, data bytes etc.) are stored. During the initialization of the device, the host CPU defines which messages are to be sent and which are to be received. Only if the CAN controller receives a message whose identifier matches with one of the identifiers of the programmed (receive-) message objects the message is stored, and the host CPU is informed by interrupt. Another advantage is that incoming Remote Frames can be answered automatically by the Full-CAN controller with the corresponding Data Frame. In this way, the CPU load is strongly reduced compared to the Basic-CAN solution. Using Full CAN devices, high baud rates and high bus loads with many messages can be handled.

•Many Full-CAN controller provide a "Basic-CAN-Feature": One of their message objects behaves like a Basic-CAN Receive Buffer, i.e. it can be programmed in a way that every message is stored there that does not match with one of the other message objects. This can be very helpful in applications where the number of message objects is not enough to receive all desired messages.