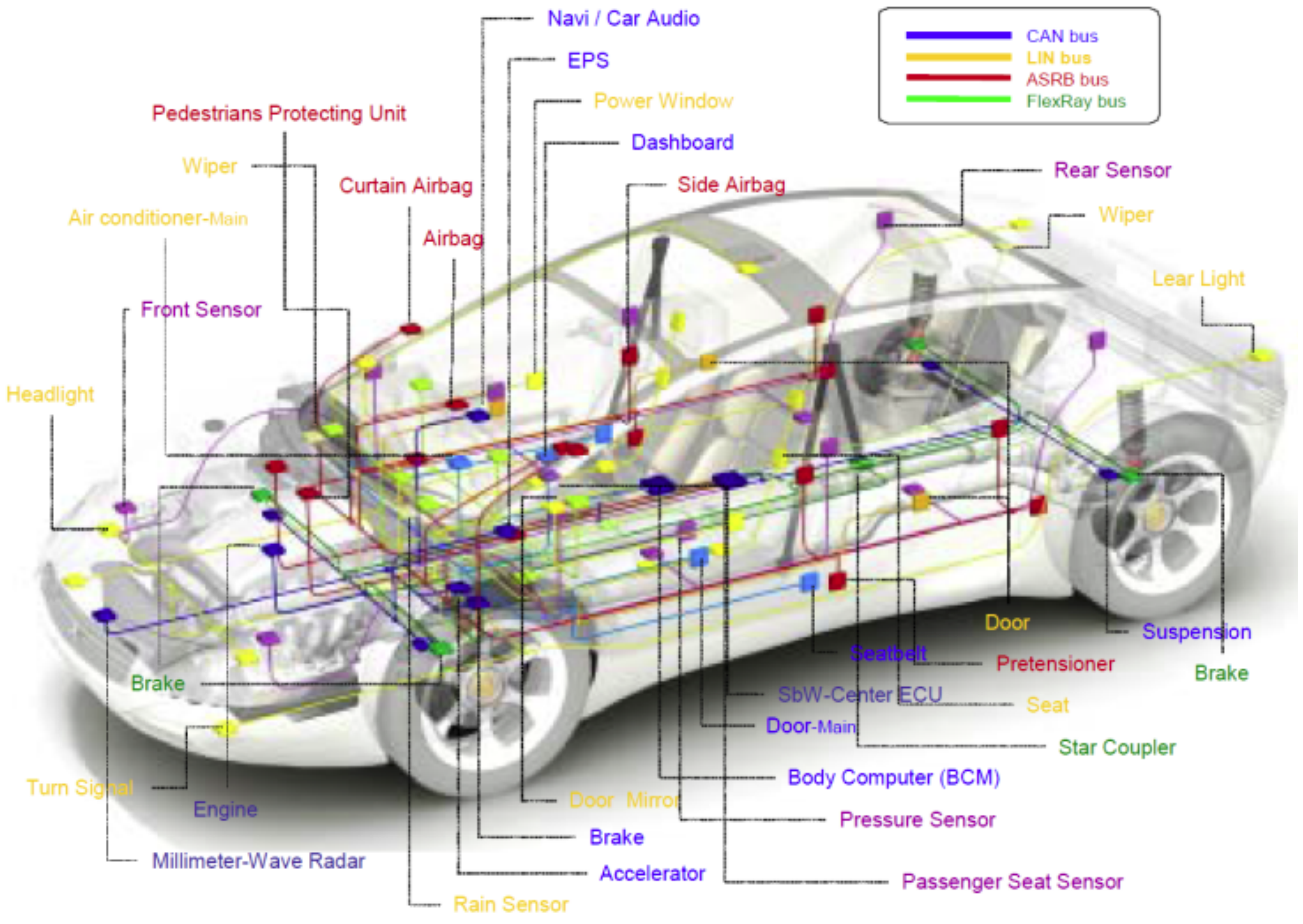
AUTOMOTIVE BASICS

Just a collective information..

CAN

The Controller Area Network (CAN) protocol incorporates a powerful means of seamlessly preventing data corruption during message collision. This arbitration process and its relationship to the electrical layer variables are explained. Techniques to force message collision and test arbitration are demonstrated with strategies to leverage arbitration as a quantitative benchmark in safety-critical systems. The benchmark is then applied to several example systems and results provided for comparison.



Introduction

The ability of a Controller Area Network to manage message collision provides a unique proving ground for protocol compliance in any application. A means of determining a benchmark for a system's performance by measuring a network's ability to execute proper arbitration is developed in this example. It is demonstrated that while a CAN bus appears to be functioning normally, many arbitration errors may be unnoticed by system operators.

CAN Implementations:
In CAN there are two main Hardware Implementations, they are Basic CAN and Full CAN.

Basic CAN: Basic CAN has only one Message buffer for Receive and Transmit messages. The received message is accepted or ignored after acceptance filtering. The decision to process a message or to ignore it is also achieved by acceptance filtering. This acceptance filtering of the node is done by software in Basic CAN. To reduce the software load at the nodes, there is a possibility to ignore some messages by ignoring specific identifiers. This is realized by bit mask for the message identifiers.

Full CAN: In Full CAN, there are 8 to 16 memory buffers for every transmitted or received message. Here the acceptance filtering is done by hardware and not by the software. Every buffer can be configured to accept messages with specific ID's.
Since the acceptance filtering is done by hardware, the software load is greatly reduced. With different buffers for different messages ensures more time for the processing of the received messages and the transmitted message can be handled according to the priority levels. Configuring each buffer for every message ensures also the data consistency in Full CAN.
Arbitration Basics

Since any CAN node may begin to transmit when the bus is free, two or more nodes may begin to transmit simultaneously. Arbitration is the process by which these nodes battle for control of the bus. Proper arbitration is critical to CAN performance because this is the mechanism that guarantees that message collisions do not reduce bandwidth or cause messages to be lost. Each data or remote frame begins with an identifier, which assigns the priority and content of the message. As the identifier is broadcast, each transmitting node compares the value received on the bus to the value being broadcast. The higher priority message during a collision has a dominant bit earlier in the identifier. Therefore, if a transmitting node senses a dominant bit on the bus in place of the recessive bit it transmitted, it interprets this as another message with higher priority transmitting simultaneously. This node suspends transmission before the next bit and automatically retransmits when the bus is idle. The result of proper arbitration is that a high-priority message transmitted without interruption is followed immediately by a low-priority message, unless of course, another high-priority message attempts to broadcast immediately following the same message. Since no messages are lost or corrupted in the collision, data and bandwidth are not compromised.

Electrical-Layer Variables (bit timing requirements)

Each CAN bit is divided into four segments (see Figure 1). The first segment, the synchronization segment (SYNC_SEG), is the time that a recessive-to-dominant or dominant-to-recessive transition is expected to occur. The second segment, the propagation time segment (PROP_SEG), is designed to compensate for the physical delay times of the network as shown in Figure 2, and should be twice the sum of the propagation delay of the bus, the input comparator delay, and the output driver delay. The third and fourth segments, both phase buffer segments (PHASE_SEG1 & PHASE_SEG2), are used for resynchronization. The bit value is sampled immediately following PHASE_SEG1.

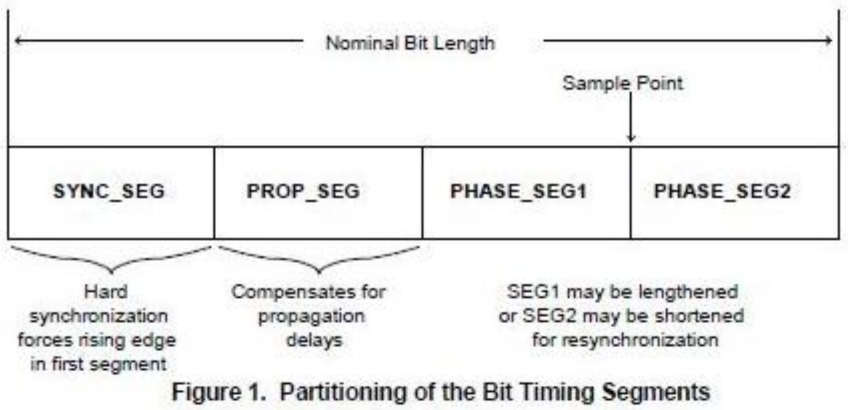

Nominal Bit Length
Sample Point
SYNC_SEG
PROP_SEG
PHASE_SEG1
PHASE_SEG2
Hard synchronization forces rising edge in first segment
Compensates for propagation delays
SEG1 may be lengthened or SEG2 may be shortened for resynchronization

Figure 1. Partitioning of the Bit Timing Segments

The bit rate may be changed by either changing the oscillator frequency, which is usually restricted by the processor requirements, or by specifying the length of the bit segments in "time quantum" and the prescaler value. The prescaler value is multiplied by the minimum time quantum, which is the reciprocal of the system clock frequency, 1/f sys , to determine the length of a working time quantum. Bit time may then be calculated as the sum of each bit segment, and the bit rate may be calculated as the reciprocal of this sum.

Each node must perform a hard synchronization upon every recessive-to-dominant edge after a bus idle or received start of frame. Hard synchronization is a restarting of the internal bit timing to force the edge into the SYNC_SEG, where edges are expected to occur. Resynchronization is performed on all other recessive-to-dominant edges of other received bits by lengthening or shortening the PHASE_SEG1 or PHASE_SEG2 by one to four time quanta as specified by the resynchronization jump width. If the difference between the edge causing resynchronization and the SYNC_SEG exceeds the resynchronization jump width, the effective result is the same as a hard synchronization.

CAN Network Errors

CAN protocol specifies five different types of network errors. A transmitting node detects a bit error when it monitors a bit value different than it is transmitting; the reaction to this condition varies with the nature of the error. A stuff error occurs when the bit-stuffing rule is violated – a bit of opposite value must be inserted immediately following any series of five consecutive bits of the same value in a message. A cyclic redundancy check (CRC) error occurs when a receiving node receives a different CRC sequence than anticipated. (Note that all nodes independently calculate the CRC sequence from the data field). A form error occurs when a field contains an illegal bit value. Finally, an acknowledgement (ACK) error occurs when the transmitter does not monitor a dominant bit in the ACK slot to signify that the message had been received properly by another node as shown in Figure 2. When a node detects a bus error, it transmits an error frame consisting of six dominant bits followed by eight recessive bits. Multiple nodes transmitting an error frame will not cause a problem because the first recessive bits will be overwritten. The result will remain six dominant bits followed by eight recessive bits, and cause the bus to be safely reset before normal communications recommence.

The CAN protocol provides a means of fault confinement by requiring each node to maintain separate receive and transmit error counters. Either counter will be incremented by 1 or 8, depending on the type of error and conditions surrounding the error. The receive error counter is incremented for errors during message reception, and the transmit error counter is incremented for errors during message transmission (for further details, see reference 1). When either of these counters exceed 127, the node is declared "error-passive," which limits the node from sending any further dominant error frames. When the transmitted error count exceeds 255, the node is declared "bus-off," which restricts the node from sending any further transmissions. The receive and transmit error counters are also decremented by 1 each time a message is received or transmitted without error, respectively. This allows a node to return from error-passive mode to error-active mode (normal transmission mode) when both counters are less than 128. The node may also return to error-active mode from bus-off mode after having received 128 occurrences of 11 consecutive recessive bits. Overall, a network maintains constant transmit and receive error counters if it averages eight properly transmitted or received messages for each error that occurs during transmission or reception, respectively.

Controller Area Network (CAN) started life in 1983 at Robert Bosch GmbH as a serial data bus standard for the interconnection of microcontrollers in vehicles. Although originally designed specifically for automotive applications, it is now also used in other applications. The protocol was officially released in 1986, and the first CAN controller chips, produced by Intel and Phillips, were available commercially in 1987. The CAN 2.0 specification was published by Bosch in 1991. The data link and physical layers of CAN for data rates of up to 125 kbps (described as "low-speed serial data communication" were defined in part two of the original ISO standard published in 1994 (ISO 11519). Part 1 of a later ISO standard published in 2003 (ISO 11898) covers the data link and physical layers of CAN, but for data rates of up to 1 Mbps. There are also a number of other related standards. The higher layer protocols used with CAN depend on the application. A number of microcontrollers (for example, Microchip Technology's PIC Microcontrollers) now have CAN support built-in.

A modern car will typically have in the order of fifty (and sometimes a lot more) *electronic control units* (ECUs) controlling various automotive sub-systems. The largest microprocessor unit in a car is usually the engine control unit (also, confusingly, commonly abbreviated to ECU). Other microprocessors control elements ranging from the transmission system and braking system, right down to cosmetic elements such in-car audio systems, and driving mirror adjustment. Some of these subsystems operate independently, but others need to communicate with each other and process and respond to data received from sensors. The CAN bus in a vehicle control system will typically connect the engine control unit with the transmission control system, for example. It is also highly suited to use as a fieldbus in general automation environments, and has become widely used for such applications, in part because of the low cost, small size and availability of many CAN controllers and processors. In automotive systems, they are an ideal alternative to expensive, cumbersome and unreliable wiring looms and connectors.

A CAN network interconnects control devices, sensors and actuators (collectively referred to here as *nodes*). Every node attached to a CAN bus can send and receive data, but not at the same time. A message consists primarily of an *identifier* that identifies the type and sender of the message, and up to eight bytes of actual data. The physical medium in a CAN network is a differential two-wire bus (usually either unshielded or shielded twisted pair), and the signaling scheme used is *Non-Return to Zero* (NRZ) with bit stuffing. Because CAN is essentially a broadcast network, messages will be received by all nodes. The messages do not reach the devices directly, but via each node?s *host-processor* and *CAN Controller*. These elements sit between the node itself and the data bus. Any node may transmit a message providing the bus is free. If two or more nodes transmit at the same time, the system of arbitration is simply to give priority based on message ID number. The message with the higher priority ID will overwrite all other messages, and any nodes responsible for the lower priority messages will back off and wait before retransmitting.

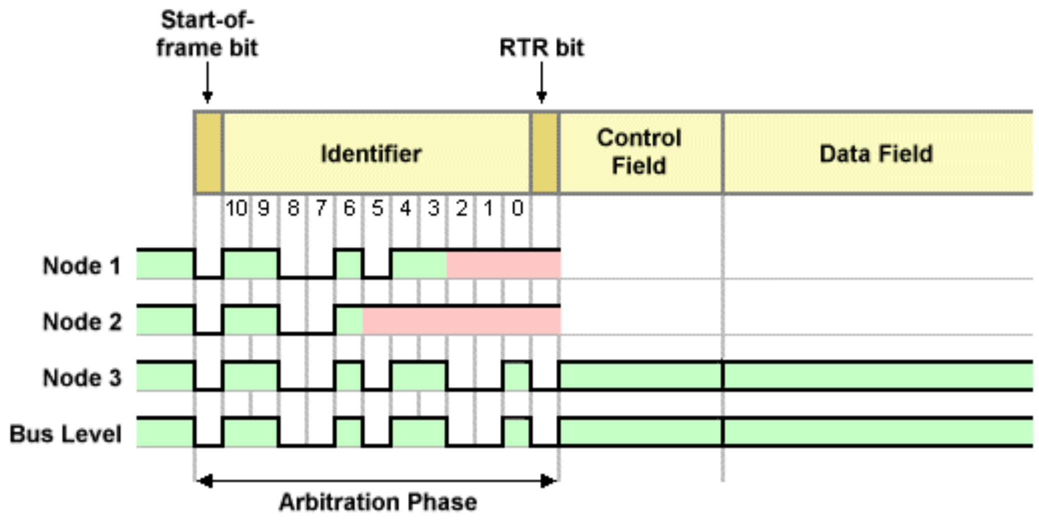
Each node will have a *host-processor* that interprets incoming messages and determines when it needs to send outgoing messages, *sensors*, *actuators* and *control devices*, which can be connected to the host-processor as required, and a *CAN Controller* which is implemented in hardware and has a synchronous clock. The CAN controller buffers incoming messages until they can be retrieved by the host-processor, generating an interrupt to let the host processor know that a message is waiting. The CAN Controller is also the buffer for outgoing messages, which it receives from the host-processor and then transmits via the bus. A transceiver handles message processing, and is usually integrated into the CAN Controller. The data rates possible are dependent on the length of the bus. Data rates of up to 1 Mbps are possible at network lengths below 40 metres. Decreasing the data rate to 125 kbps would allow a network length of up to 500 metres.

Transmission of messages in a CAN is based on the *producer-consumer* (broadcast) principle.

A message transmitted by one node (the *producer*) is received by all other nodes (the *consumers*). Messages do not have a destination address, but a *Message ID*. Messages in the standard format have an 11-bit Message ID, enabling 2,048 different messages to be defined for any one system – more than sufficient for most applications. For applications that require a larger number of messages, an extended message format with a 29-bit Message ID may be used, allowing over five hundred million different messages to be defined. Only certain messages will apply to each node on the network, so a node receiving a message must apply *acceptance filtering* (usually implemented in hardware, and based on the Message ID). If the message received by a node is relevant to it, it will be processed, otherwise it will be ignored. CAN networks may be expanded without modification to existing hardware or software if the devices to be added are purely receivers, and if they only require messages that are already generated by the network.

Arbitration in CAN networks

The standard form of arbitration in a CAN network is *Carrier Sense Multiple Access/Bitwise Arbitration* (CSMA/BA). If two or more nodes start transmitting at the same time, arbitration is based on the priority level of the message ID, and allows the message whose ID has the highest priority to be delivered immediately, without delay. This makes CAN ideal for real-time, priority-based systems. Each node, when it starts to transmit its Message ID, will monitor the bus state and compare each bit received from the bus with the bit transmitted. If a *dominant bit* (0) is received when a *recessive bit* (1) has been transmitted, the node stops transmitting because another node has established priority. The concept is illustrated by the diagram below.



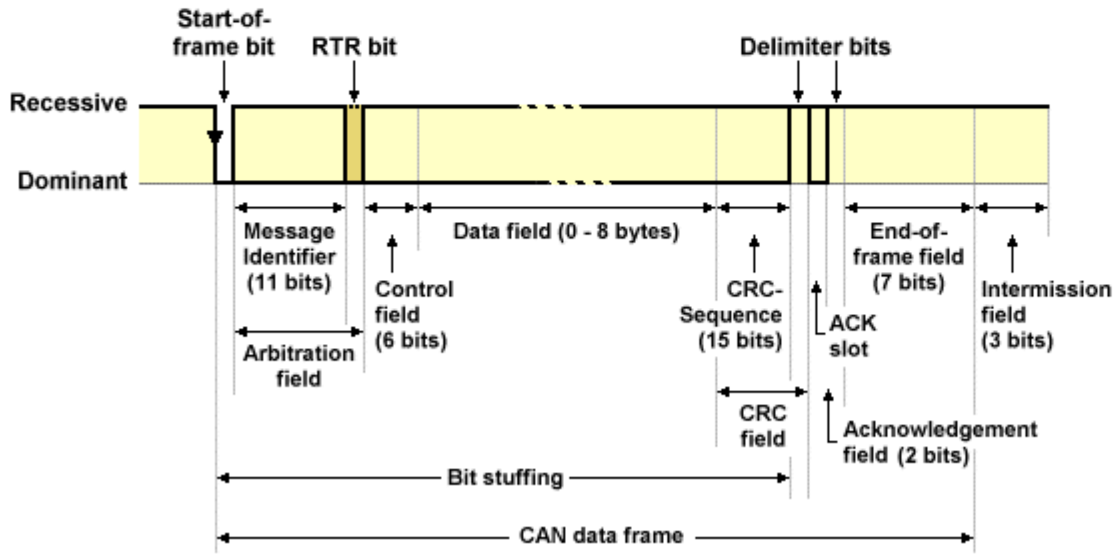
Bitwise arbitration in CAN networks

Arbitration is performed as the identifier field is transmitted, and is *non-destructive*. Each node transmits its 11-bit Message ID, starting with the highest-order bit (bit 10). Binary zero (0) is a *dominant bit*, and binary one (1) is a *recessive bit*. Because a dominant bit will overwrite a recessive bit on the bus, the state of the bus will always reflect the state of the message ID with the highest priority (i.e. the lowest number). As soon as a node sees a bit comparison that is unfavourable to itself, it will cease to participate in the arbitration process and wait until the bus is free again before attempting to retransmit its message. The message with the highest priority will thus continue to be transmitted without delay, and unimpeded. In the above illustration, Node 2 transmits bit 5 as a recessive bit (1), while the bus level read is dominant (0), so Node 2 will back off. Similarly, Node 1 will back off after transmitting bit 2 as a recessive bit, whereas the bus level remains dominant. Node 3 is then free to complete transmission of its message.

The Message ID for each system element is assigned by the system designer, and the arbitration method used ensures that the highest-priority messages will always be transmitted ahead of another message, should simultaneous transmissions occur. The bus is thus allocated on the basis of need. The only limiting factor is therefore the capacity of the bus itself. Outstanding transmission requests are dealt with in their order of priority, with minimum delay and maximum utilisation of the available bus capacity. In any system, some parameters will change more rapidly than others. In a motor vehicle, for example, the rpm of the engine will change far more rapidly than the temperature of the engine coolant. The more rapidly changing parameters are probably going to need more frequent monitoring, and for this reason will probably be given a higher priority.

CAN Frame Format

The general format of a CAN message frame is shown below.



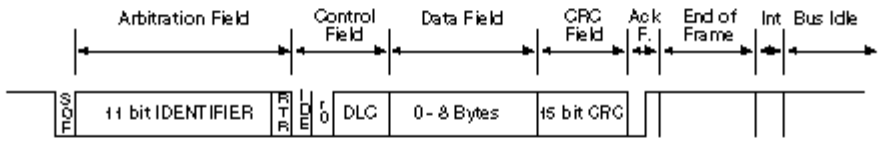
Data is transmitted using *Message Frames*. The standard CAN protocol (version 2.0A), also known as *Base Frame Format*, uses an 11-bit Message ID. The extended CAN protocol (version 2.0B), also now known as *Extended Frame Format*, supports both 11-bit and 29-bit Message IDs. Most version 2.0A controllers are tolerant of extended format messages, but essentially ignore them. Version 2.0B controllers can send and receive messages in both formats.
The start of a message frame is signaled by a dominant *start-of-frame* bit, followed by the 11-bit *Message ID* and the *Remote Transmission Request* (RTR) bit, which is only set if the message is a *data request frame* (as opposed to a data frame). It should probably be noted here that, although nodes on a CAN network generally send data without being polled, a node may request the transmission of a specific message by another node in the system. The first two bits (*r0* and *r1*) of the 6-bit control field specify the transmission format (i.e. *standard* or *extended*), while the last four bits form the *Data Length Code* (DLC), which indicates the number of bytes of data transmitted. The *data* field can contain from zero to eight bytes of

data, and is followed by the 16-bit *CRC* field, containing a 15-bit cyclic redundancy check code which is used by the receiving node to detect errors, and a recessive *delimiter bit*.

The *ACKnowledge* field has two bits. The first is the *ACK Slot* which is transmitted as a recessive bit, but will be overwritten with a dominant bit by any node that successfully receives the transmitted message. The second bit is a recessive *delimiter bit*. The *end-of-frame* field consists of seven recessive bits, and signals that error-free transmission of the message has been completed. The end-of-frame field is followed by the *intermission* field consisting of three recessive bits, after which the bus may be considered to be free for use. Idle time on the bus may be of any length, including zero.

At a data rate of 1 Mbps, it is possible to send in the order of ten thousand standard format messages per second over a CAN network, assuming an average data length of four bytes. The number of messages that could be sent would come down to around seven thousand if all the messages contained the full eight bytes of data allowed. One of the major benefits of CAN is that, if several controllers require the same data from the same device, only one sensor is required rather than each controller being connected to a separate sensor. As mentioned previously, the data rate that can be achieved is dependent on the length of the bus, since the bit time interval is adjusted upwards to compensate for any increase in the time required for signals to propagate along the bus, which is proportional to the length of the bus. Bus length and bit rate are thus inversely proportional.

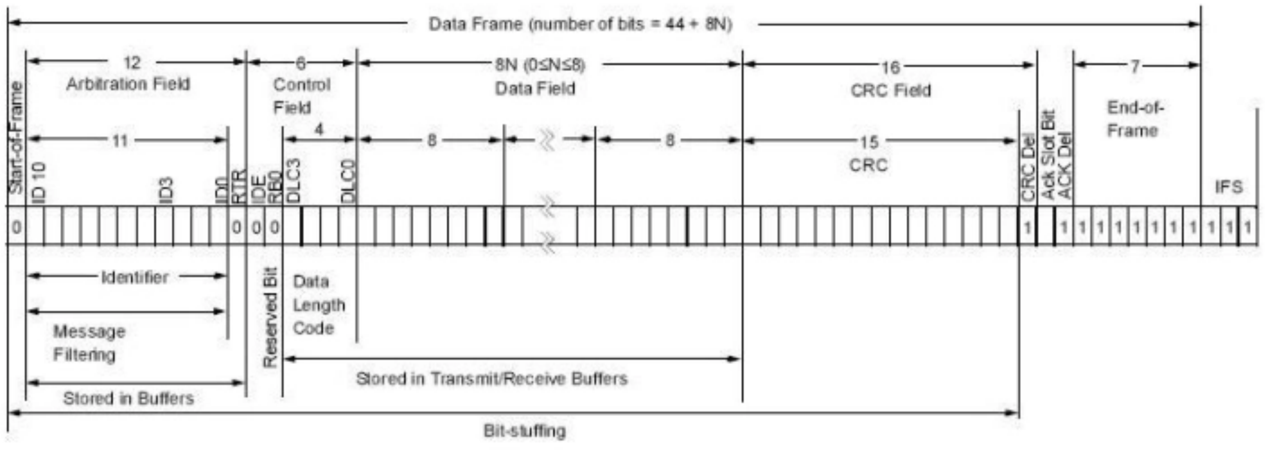
Message frame format



Message frame for standard format (CAN Specification 2.0A)
The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier (ID). In the standard format the length of the ID is 11 bits and in the extended format the length is 29 bits. The message frame for transmitting messages on the bus comprises seven main fields.

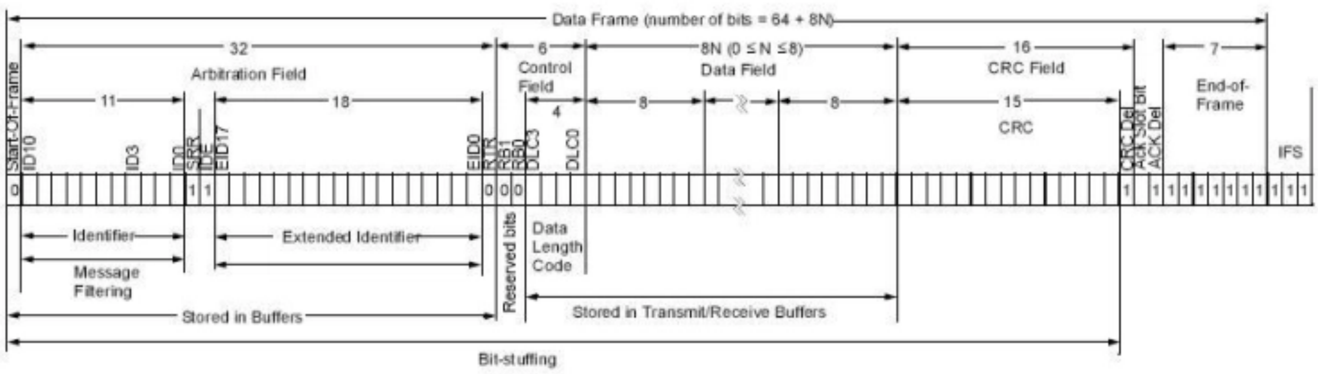
A message in the standard format begins with the start bit “start of frame”, this is followed by the “arbitration field”, which contains the identifier and the “RTR” (remote transmission request) bit, which indicates whether it is a data frame or a request frame without any data bytes (remote frame). The “control field” contains the IDE (identifier extension) bit, which indicates either standard format or extended format, a bit reserved for future extensions and – in the last 4 bits – a count of the data bytes in the data field. The “data field” ranges from 0 to 8 bytes in length and is followed by the “CRC field”, which is used as a frame security check for detecting bit errors. The “ACK field” comprises the ACK slot (1 bit) and the ACK delimiter (1 recessive bit). The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by those receivers which have at this time received the data correctly (positive acknowledgement). Correct messages are acknowledged by the receivers regardless of the result of the acceptance test. The end of the message is indicated by “end of frame”. “Intermission” is the minimum number of bit periods separating consecutive messages. If there is no following bus access by any station, the bus remains idle (“bus idle”).

Standard Data Frame



The CAN standard data frame is shown in Figure 2-1. As with all other frames, the frame begins with a Start- Of-Frame (SOF) bit, which is of the dominant state and allows hard synchronization of all nodes. The SOF is followed by the arbitration field, consisting of 12 bits: the 11-bit identifier and the Remote Transmission Request (RTR) bit. The RTR bit is used to distinguish a data frame (RTR bit dominant) from a remote frame (RTR bit recessive). Following the arbitration field is the control field, consisting of six bits. The first bit of this field is the Identifier Extension (IDE) bit, which must be dominant to specify a standard frame. The following bit, Reserved Bit Zero (RB0), is reserved and is defined as a dominant bit by the CAN protocol. The remaining four bits of the control field are the Data Length Code (DLC), which specifies the number of bytes of data (0 – 8 bytes) contained in the message. After the control field is the data field, which contains any data bytes that are being sent, and is of the length
defined by the DLC (0 – 8 bytes). The Cyclic Redundancy Check (CRC) field follows the data field and is used to detect transmission errors. The
CRC field consists of a 15-bit CRC sequence, followed by the recessive CRC Delimiter bit. The final field is the two-bit Acknowledge (ACK) field.
During the ACK Slot bit, the transmitting node sends out a recessive bit. Any node that has received an error-free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not). The recessive acknowledge delimiter completes the acknowledge field and may not be overwritten by a dominant bit.

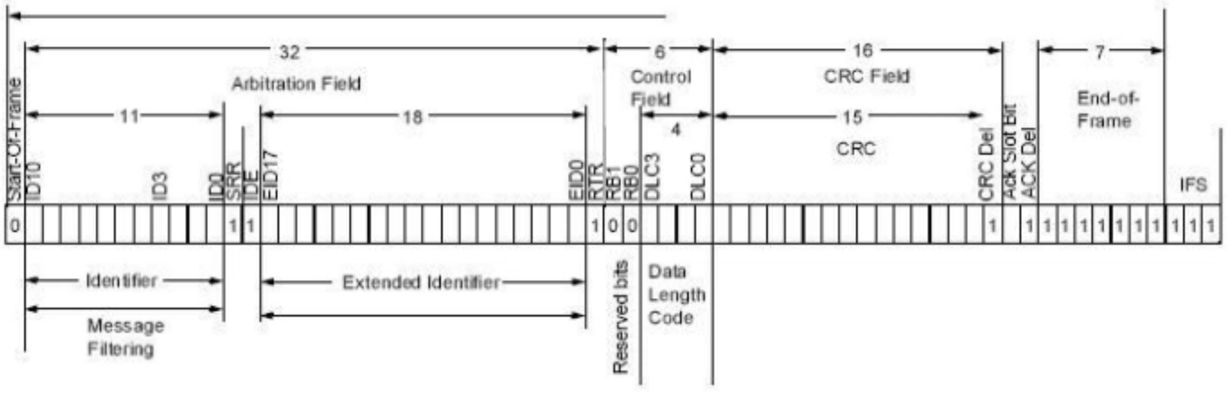
Extended Data Frame



In the extended CAN data frame, shown in Figure 2-2, the SOF bit is followed by the

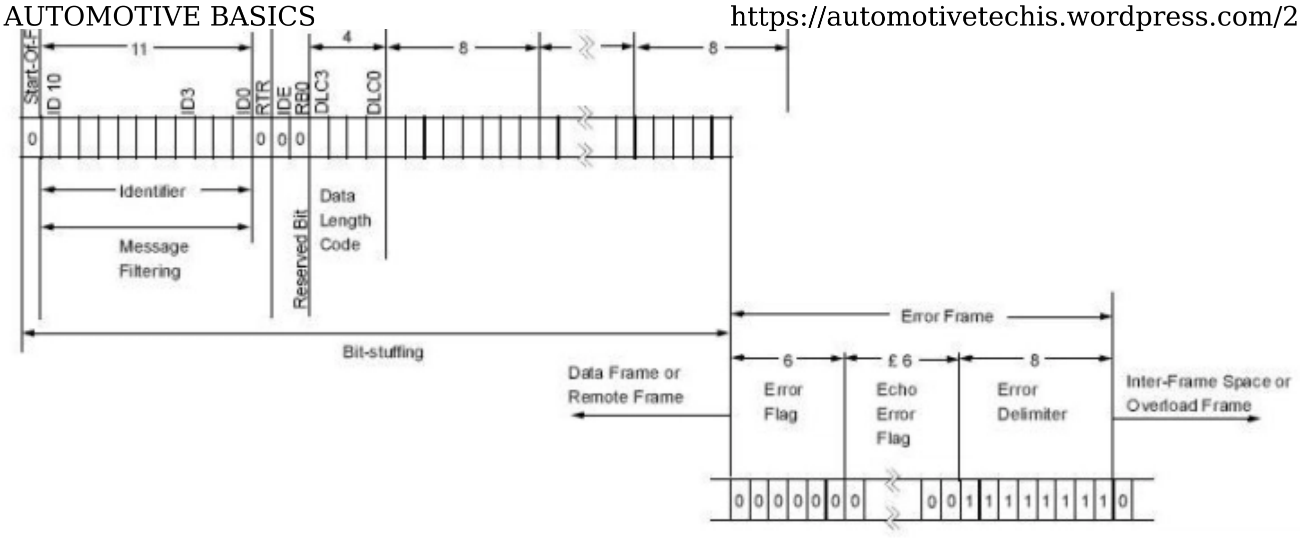
arbitration field, which consists of 32 bits. The first 11 bits are the Most Significant bits (MSb) (Base-lD) of the 29-bit identifier. These 11 bits are followed by the Substitute Remote Request (SRR) bit, which is defined to be recessive. The SRR bit is followed by the lDE bit, which is recessive to denote an extended CAN frame. It should be noted that if arbitration remains unresolved after transmission of the first 11 bits of the identifier, and one of the nodes involved in the arbitration is
sending a standard CAN frame (11-bit identifier), the standard CAN frame will win arbitration due to the assertion of a dominant lDE bit. Also, the SRR bit in an extended CAN frame must be recessive to allow the assertion of a dominant RTR bit by a node that is sending a standard CAN remote frame. The SRR and lDE bits are followed by the remaining 18 bits of the identifier (Extended lD) and the remote transmission request bit.
To enable standard and extended frames to be sent across a shared network, the 29-bit extended message identifier is split into 11-bit (most significant) and 18-bit (least significant) sections. This split ensures that the lDE bit can remain at the same bit position in both the standard and extended frames. Following the arbitration field is the six-bit control field. The first two bits of this field are reserved and must be
dominant. The remaining four bits of the control field are the DLC, which specifies the number of data bytes contained in the message.
The remaining portion of the frame (data field, CRC field, acknowledge field, end-of-frame and intermission) is constructed in the same way as a standard data frame

Remote Data Frame



Normally, data transmission is performed on an autonomous basis by the data source node (e.g., a sensor sending out a data frame). It is possible,
however, for a destination node to request data from the source. To accomplish this, the destination node sends a remote frame with an identifier that matches the identifier of the required data frame. The appropriate data source node will then send a data frame in response to the remote frame request. There are two differences between a remote frame (shown in Figure 2-3) and a data frame. First, the RTR bit is at the recessive state and, second, there is no data field. In the event of a data frame and a remote frame with the same identifier being transmitted at the
same time, the data frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the remote frame receives the desired data immediately.

Interrupted Data Frame
12
Arbitration Field
6
Control Field
8N (0≤N≤8)
Data Field



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

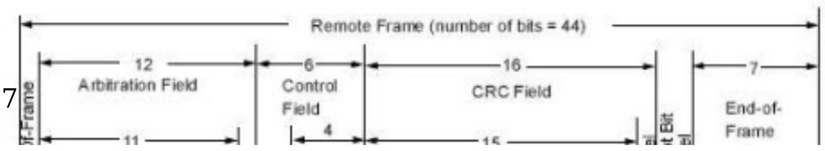
2.4.1 ACTIVE ERRORS
If an error-active node detects a bus error, the node interrupts transmission of the current message by generating an active error flag. The active error flag is composed of six consecutive dominant bits. This bit sequence actively violates the bit-stuffing rule. All other stations recognize the resulting bit-stuffing error and, in turn, generate error frames themselves, called error echo flags.

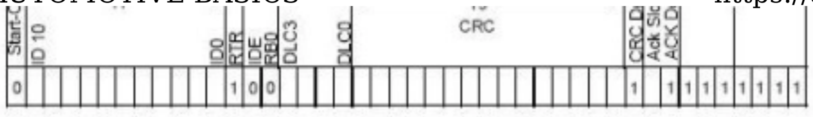
The error flag field, therefore, consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The error delimiter field (eight recessive bits) completes the error frame. Upon completion of the error frame, bus activity returns to normal and the interrupted node attempts to resend the aborted message.

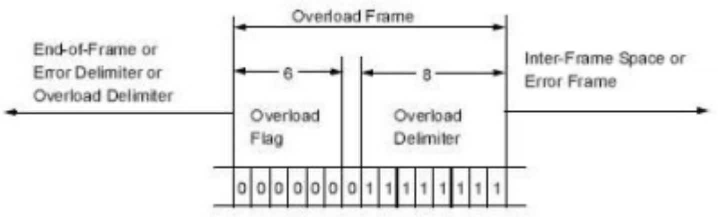
2.4.2 PASSIVE ERRORS
If an error-passive node detects a bus error, the node transmits an error-passive flag followed by the error delimiter field. The error-passive flag consists of six consecutive recessive bits. The error frame for an errorpassive node consists of 14 recessive bits. From this it follows that, unless the bus error is detected by an erroractive node or the transmitting node, the message will continue transmission because the error-passive flag does not interfere with the bus.

If the transmitting node generates an error-passive flag, it will cause other nodes to generate error frames due to the resulting bit-stuffing violation. After transmission of an error frame, an error-passive node must wait for six consecutive recessive bits on the bus before attempting to rejoin bus communications. The error delimiter consists of eight recessive bits and allows the bus nodes to restart bus communications cleanly after an error has occurred.

Overload Frame,Interframe Space







An overload frame, shown in Figure 2-5, has the same format as an active error frame. An overload frame, however, can only be generated during an interframe space. In this way, an overload frame can be differentiated from an error frame (an error frame is sent during the transmission of a message). The overload frame consists of two fields: an overload flag followed by an overload delimiter. The overload flag consists of six dominant bits followed by overload flags generated by other nodes (and, as for an active error flag, giving a maximum of twelve dominant bits). The overload delimiter consists of eight recessive bits. An overload frame can be generated by a node as a result of two conditions:
1. The node detects a dominant bit during the interframe space, an illegal condition. Exception: The dominant bit is detected during the third bit of IFS. In this case, the receivers will interpret this as a SOF.

2. Due to internal conditions, the node is not yet able to begin reception of the next message. A node may generate a maximum of two sequential overload frames to delay the start of the next message.**CAN BUS MESSAGE FRAMES – Interframe Space**The interframe space separates a preceding frame (of any type) from a subsequent data or remote frame. The interframe space is composed of at least three recessive bits called the Intermission. This allows nodes time for internal processing before the start of the next message frame. After the intermission, the bus line remains in the recessive state (bus idle) until the next transmission starts.
Detecting and signalling errors

Unlike other bus systems, the CAN protocol does not use acknowledgement messages but instead signals any errors that occur.

Cyclic Redundancy Check (CRC)

The CRC safeguards the information in the frame by adding redundant check bits at the transmission end. At the receiver end these bits are re-computed and tested against the received bits. If they do not agree there has been a CRC error.

Frame check

This mechanism verifies the structure of the transmitted frame by checking the bit fields against the fixed format and the frame size. Errors detected by frame checks are designated “format errors”.

ACK errors

As mentioned above, frames received are acknowledged by all recipients through positive acknowledgement. If no acknowledgement is received by the transmitter of the message (ACK error) this may imply that there is a transmission error which has been detected only by the recipients, that the ACK field has been corrupted or that there are no receivers.

The CAN protocol also implements two mechanisms for error detection at the bit level:

Monitoring

The ability of the transmitter to detect errors is based on the monitoring of bus signals: each node which transmits also observes the bus level and thus detects differences between the bit sent and the bit received. This permits reliable detection of all global errors and errors local to the transmitter.

Bit stuffing

The coding of the individual bits is tested at bit level. The bit representation used by CAN is NRZ (non-return-to-zero) coding, which guarantees maximum efficiency in bit coding. The synchronization edges are generated by means of bit stuffing, i.e. after five consecutive equal bits the sender inserts into the bit stream a stuff bit with the complementary value, which is removed by the receivers. The code check is limited to checking adherence to the stuffing rule.

If one or more errors are discovered by at least one station (any station) using the above mechanisms, the current transmission is aborted by sending an "error flag". This prevents other stations accepting the message and thus ensures the consistency of data throughout in the network.
After transmission of an erroneous message has been aborted, the sender automatically re-attempts transmission (automatic repeat request). There may again be competition for bus allocation. As a rule, retransmission will be begun within 23 bit periods after error detection; in special cases the system recovery time is 31 bit periods.

However effective and efficient the described method may be, in the event of a defective station it might lead to all messages (including correct ones) being aborted, thus blocking the bus system if no measures for self-monitoring were taken. The CAN protocol therefore provides a mechanism for distinguishing sporadic errors from permanent errors and localizing station failures (fault confinement).

This is done by statistical assessment of station error situations with the aim of recognizing a station's own defects and possibly entering an operating mode where the rest of the CAN network is not negatively affected. This may go as far as the station switching itself off to prevent messages erroneously recognized as incorrect from being aborted.

Extended format CAN messages

The SAE "Truck and Bus" subcommittee standardized signals and messages as well as data transmission protocols for various data rates. It became apparent that standardization of this kind is easier to implement when a longer identification field is available.

To support these efforts, the CAN protocol was extended by the introduction of a 29-bit identifier. This identifier is made up of the existing 11-bit identifier (base ID) and an 18-bit extension (ID extension). Thus the CAN protocol allows the use of two message formats:

- StandardCAN (Version 2.0A) und
- ExtendedCAN (Version 2.0B)

As the two formats have to coexist on one bus it is laid down which message has higher priority on the bus in the case of bus access collisions with differing formats and the same base identifier:

! the message in standard always has priority over the message in extended format.

Message frame standard format (CAN specification 2.0A)
CAN controllers which support the messages in extended format can also send and receive messages in standard format. When CAN controllers which only cover the standard format (Version 2.0A) are used on one network, then only messages in standard format can be transmitted on the entire network. Messages in extended format would be misunderstood. However there are CAN controllers which only support standard format but recognize messages in extended format and ignore them (Version 2.0B passive).

The distinction between standard format and extended format is made using the IDE bit (Identifier Extension Bit) which is transmitted as dominant in the case of a frame in standard format. For frames in extended format it is recessive.

The RTR bit is transmitted dominant or recessive depending on whether data are being transmitted or whether a specific message is being requested from a station. In place of the RTR bit in standard format the SRR (substitute remote request) bit is transmitted for frames with extended ID. The SRR bit is always transmitted as recessive, to ensure that in the case of arbitration the standard frame always has priority bus allocation over an extended frame when both messages have the same base identifier.

Message frame for extended format (CAN specification 2.0B)

Unlike the standard format, in the extended format the IDE bit is followed by the 18-bit ID extension, the RTR bit and a reserved bit (r1).

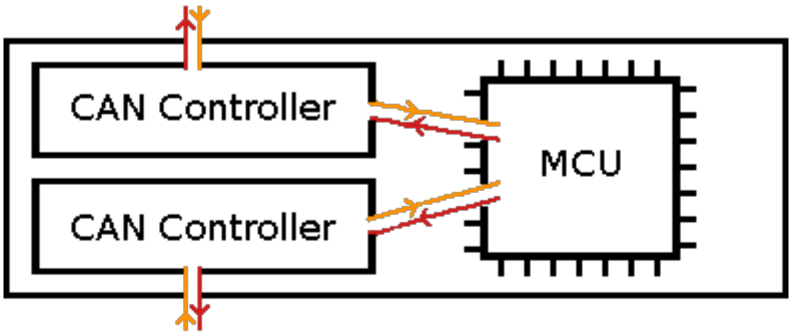
All the following fields are identical with standard format. Conformity between the two formats is ensured by the fact that the CAN controllers which support the extended format can also communicate in standard format.

Error detection and management

Nodes that transmit messages on a CAN network will monitor the bus level to detect transmission errors, which will be globally effective. In addition, nodes receiving messages will monitor them to ensure that they have the correct format throughout, as well as recalculating the CRC to detect any transmission errors that have not previously been detected (i.e. locally effective errors). The CAN protocol also has a mechanism for detecting and shutting down defective network nodes, ensuring that they cannot continually disrupt message transmission.

When errors are detected, either by the transmitting node or a receiving node, the node that detects the error signals an error condition to all other nodes on the network by transmitting an error message frame containing a series of six consecutive bits of the dominant polarity. This triggers an error, because the *bit-stuffing* used by the signalling scheme means that messages should never have more than five consecutive bits with the same polarity (when bit-stuffing is employed, the transmitter inserts a bit of opposite polarity after five consecutive bits of the same polarity. The additional bits are subsequently removed by the receiver, a process known as *de-stuffing*). All network nodes will detect the error message and discard the offending message (or parts thereof, if the whole message has not yet been received). If the transmitting node generates or receives an error message, it will immediately thereafter attempt to retransmit the message

CAN gateway



A CAN/CAN gateway incorporates two CAN controller with a microcontroller. CAN messages are received by a CAN controller, processed by the microcontroller and then sent by the opposite CAN controller. Processed means that messages may be filtered, remapped to different CAN identifiers or that the data content of the messages may be altered. Also different baud rates may be used on both sides.

A CAN/CAN gateway connects two CAN systems and controls the message exchange by applying rules and functions on these messages. This distinguishes the gateway from the repeater, which acts more or less like a piece of cable. This extended functionality leads to higher costs in respect to a CAN repeater.

The main issue while using a CAN/CAN gateway is the latency time for a received message to be sent out on the other side. For an idle network this time is the propagation delay of the gateway. But as soon as there is busload this time gets undetermined. This has two reasons. Even if there is only busload on the receiving side, the propagation delay increases, because the microcontroller has to spend more and more time in the CAN receive interrupt routine and a message to be sent out has to wait for the processor to have time for doing so. If there is busload on both sides, the CAN controller has to wait for an idle bus to get access to it. This time increases with the busload and depends also on the CAN identifier used due to their priority. Therefor the CAN system integrator has to take extra care on the busload, if a gateway is going to be used.

From the application point of view the implications of this issue get obvious while e.g.

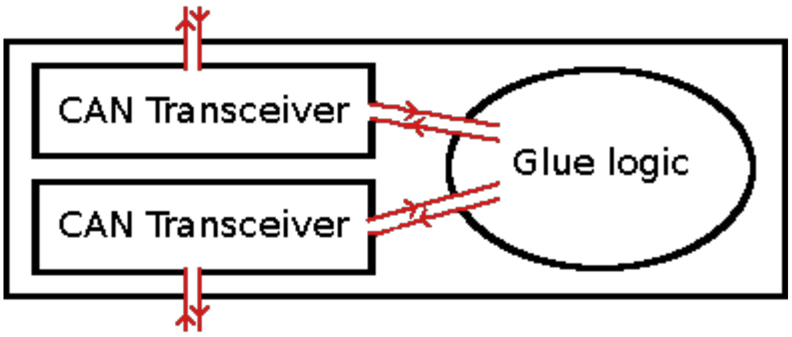
looking on the SYNC message of the CANopen protocol. This message is used to synchronize actions on different CANopen nodes to the moment they receive the message. If the SYNC message is delayed by an arbitrary period of time, system behavior tends to get unpredictable. Filtering, also by using the acceptance filter of the CAN controller will help, but cannot reduce the effect completely.

Due to the fact that the CAN/CAN gateway acts on messages, error frames will not be distributed from one side to the other. Furthermore CAN/CAN gateways can have the full extend of bus length on both sides for a given baud rate and the full count of modules.

The motivation to use a CAN/CAN gateway is given, if two CAN systems should be connected, but the message flow has to be controlled. In this case the system integrator benefits from the versatile functionality offered by this devices. While e. g. developing a new electronic control unit (ECU) for an automotive application, the system designer may use a CAN/CAN gateway to shift identifiers or to modify data contents of specific messages and is able by this means to combine old and new units.

A CAN/CAN gateway which routes the CAN messages over a media like an optical fiber (CG-FL, EtherCAN FX) or Ethernet (EtherCAN CI) using a protocol like TCP or UDP is called a CAN/CAN router. They extend the concept of a gateway to a larger physical expansion, but preserve the attributes of a gateway. This devices are able to realize point to point connections of CAN systems over distances up to 40km with low latencies and enable sophisticated CAN applications.

CAN Repeater



Basic functioning scheme of a CAN repeater
A CAN repeater incorporates two CAN transceiver with a glue logic. It propagates a CAN signal from one side to the other and vice versa. Therefor an ideal CAN repeater acts like a piece of cable, it is transparent for the CAN signal. Due to the propagation delay of the two CAN transceiver and the glue logic an equivalent length can be given for a specific CAN repeater. It is about 40m for a repeater without galvanic decoupling and about 60m for a device with galvanic decoupling.

The maximum length of a CAN system for a given baud rate cannot be extended by the use of a CAN repeater. But a CAN repeater allows to implement topologies different than a simple line. Stub lines or star topologies can be realized by using CAN repeaters.

If a CAN network is divided into two segments by a CAN repeater both of them must be terminated correctly with two 120R resistors at its ends. Both segments are physically independent, but form a single CAN network from the logical point of view. This means that the maximum length of a stub line is only determined by the maximum distance between two end points of the network. A CAN line, which has a stub somewhere in the middle, has three endpoints “A”, “B” and “C”. The maximum of the three distances “AB”, “AC” and “BC” is essential for determining the maximum baud rate in this specific system. It has to be noted that the equivalent cable length of the CAN repeater has to be taken into concern. For an example on this topic visit our article: CAN Repeater example

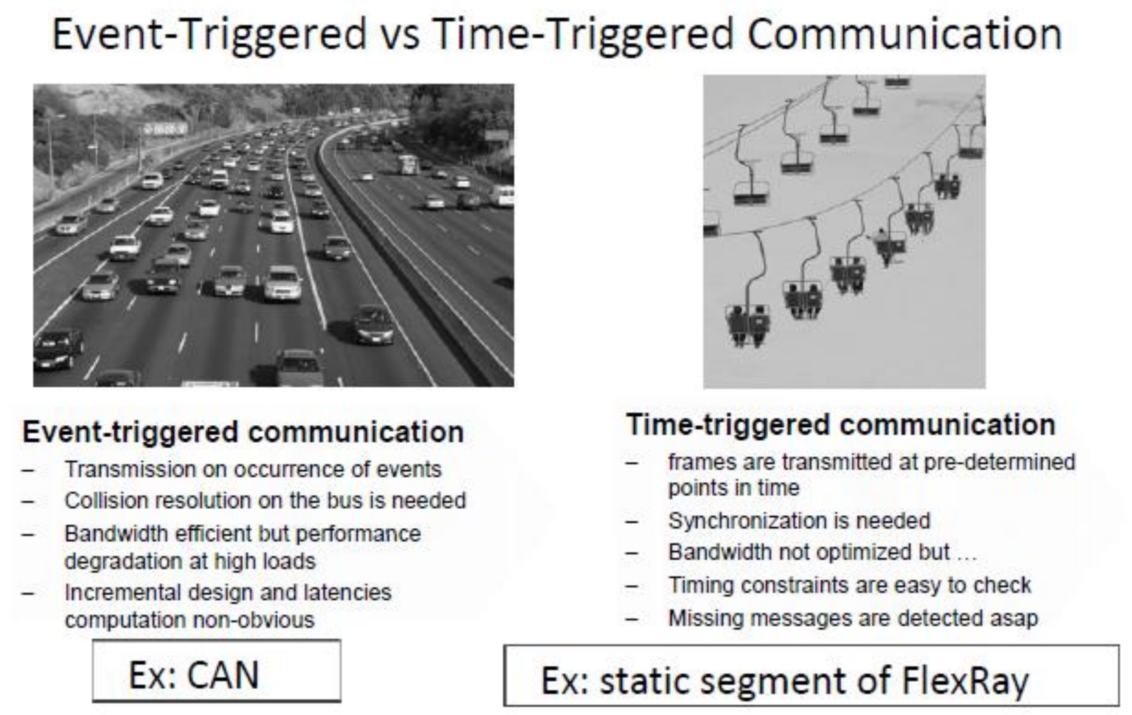
A CAN repeater can be used to regenerate the CAN signal for very long CAN lines or it can help to increase the maximum count of nodes in a CAN system. Due to the fact that a CAN repeater is transparent for the CAN signal error frames are also propagated. But a repeater may offer the functionality to disconnect a segement, which is locked to a permanent dominant state. This can help to increase the system reliability. Most of our repeaters include this feature.

Our CAN repeaters are offered with a parameter called inhibit time. It is important to choose the correct value for this parameter that the repeater will work as expected in a dedicated system. We recommend to set it to 10-20% of the bit time.

The most important motivation to use a CAN repeater is to implement a network topology which is not a line. This approach can help to decrease the overall length of a CAN system. If a galvanic decoupling between two parts of a CAN network is needed, it can be realized by a galvanic decoupled CAN repeater.

A CAN repeater gives the opportunity to offer a solution to network problems, which otherwise could only be solved by higher costs or as worst case by having to use something different than CAN.

CAN & Flexray differences:



Event-Triggered vs Time-Triggered Communication

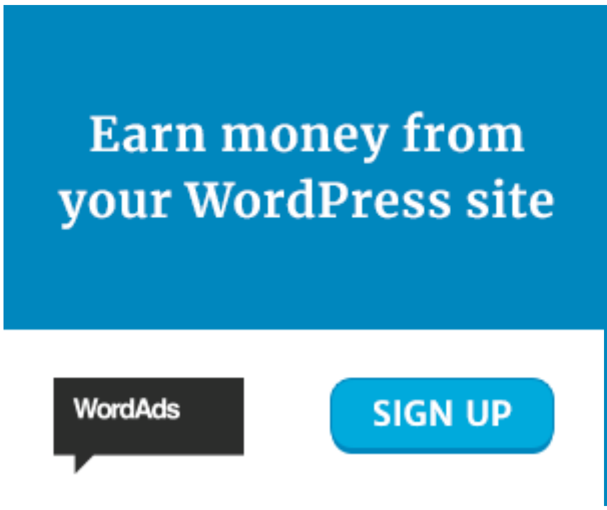
Event-triggered communication
- Transmission on occurrence of events
- Collision resolution on the bus is needed
- Bandwidth efficient but performance degradation at high loads
- Incremental design and latencies computation non-obvious

Ex: CAN

Time-triggered communication
- frames are transmitted at pre-determined points in time
- Synchronization is needed
- Bandwidth not optimized but …
- Timing constraints are easy to check
- Missing messages are detected asap

Ex: static segment of FlexRay

[youtube http://www.youtube.com/watch?v=sw3ADKPo1Uo&feature=player_detailpage]


Earn money from your WordPress site
WordAds
SIGN UP

Earn money off your WordPress site
WordAds