UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Student: Rogoz Bogdan Dorin

Group: 30433

Table of Contents

1. Theme

The topic discussed in this paper is the CAN standard used within the automotive industry.

2. Objective

The objective is to provide an overview on CAN systems' communications protocol. After finishing the paper, the reader should be able to understand the theoretical fundamentals behind the CAN 2.0A's architecture and messaging protocol.

3. Specifications

This paper serves as an entry point on the road of building a fully-fledged CAN system, a simulator or the logic behind a CAN controller linking a certain microprocessor inside an ECU to the rest of the CAN Bus. By using the information presented, the engineer can make his way through learning the more advanced specifications of the CAN standard, in order to be able to build a fully-working system (including error checking, correction, bit stuffing etc.).

4. Abstract

A Controller Area Network (CAN) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but is also used in many other contexts.

5. History

The CAN 1.0 protocol was released in 1986. The first CAN controller chips, produced by Intel and Philips, came on the market in 1987.

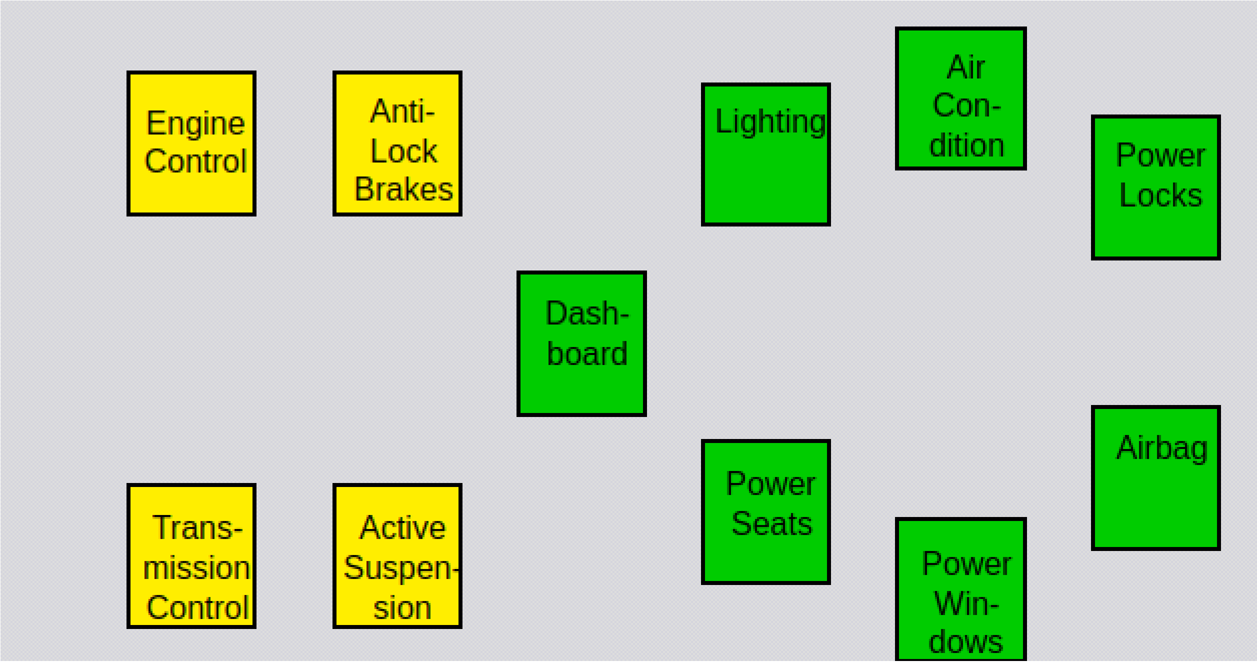
Bosch published several versions of the CAN specification and the latest is CAN 2.0 published in 1991. This specification has two parts; part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier. A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

Bosch is still active in extending the CAN standards. In 2012, Bosch released CAN FD 1.0 or CAN with Flexible Data-Rate.

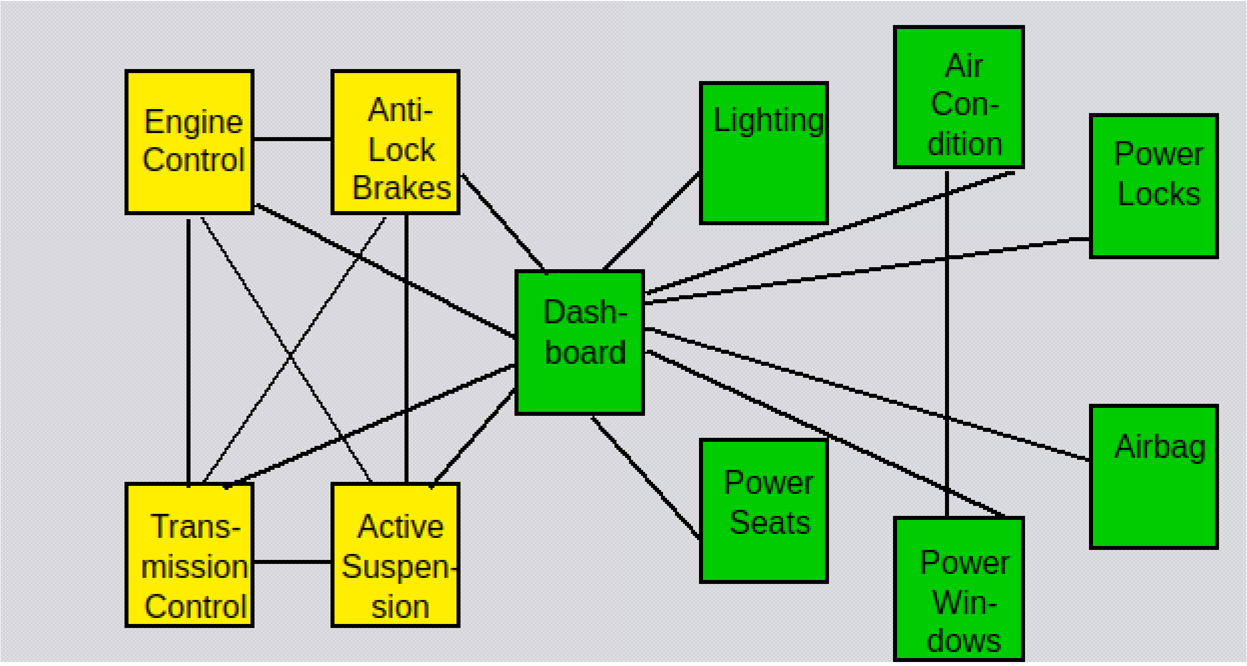
6. Applications

The modern automobile have large numbers of ECUs (Electronic Control Units) and it is essential that they are capable of intercommunicating. Before CAN systems became available, the intercommunications system occupied large amounts of space and quickly became hard to develop and test.

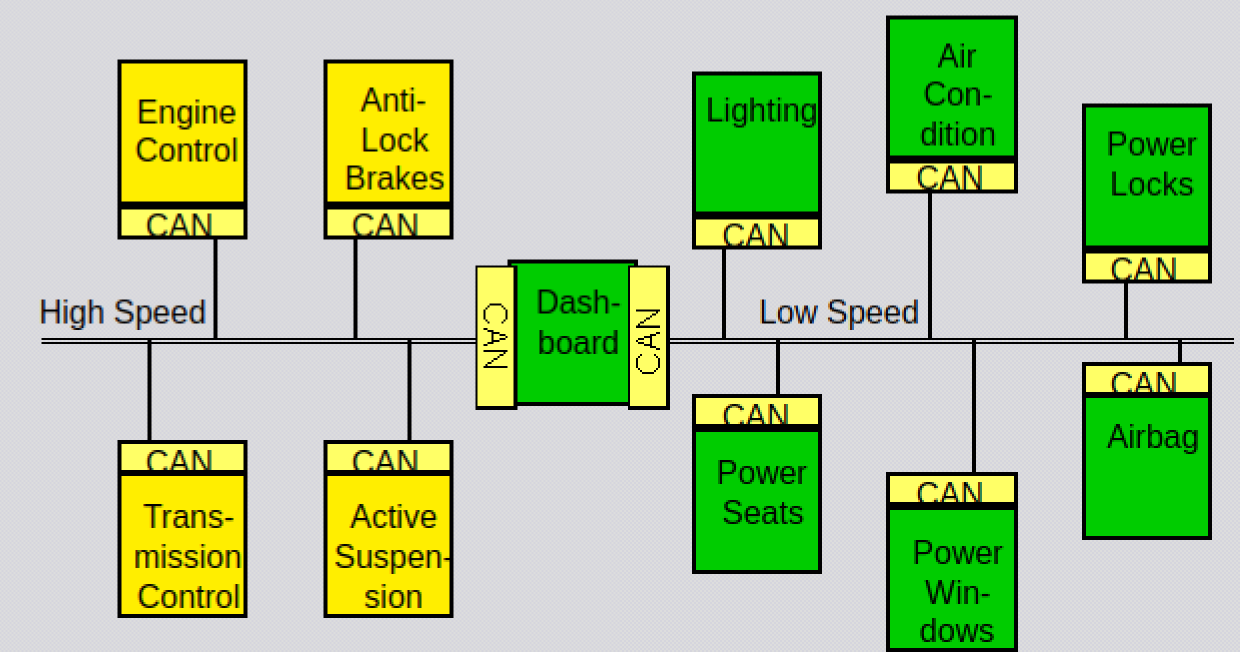
Let's give the following example: suppose we have the following ECUs on our vehicle:



Interconnecting these ECUs in a traditional way would result in this:



Of course, such wiring (point-to-point wiring) would produce high material costs, high time delays and reliability problems. On the other hand, using a CAN architecture produces the following system:

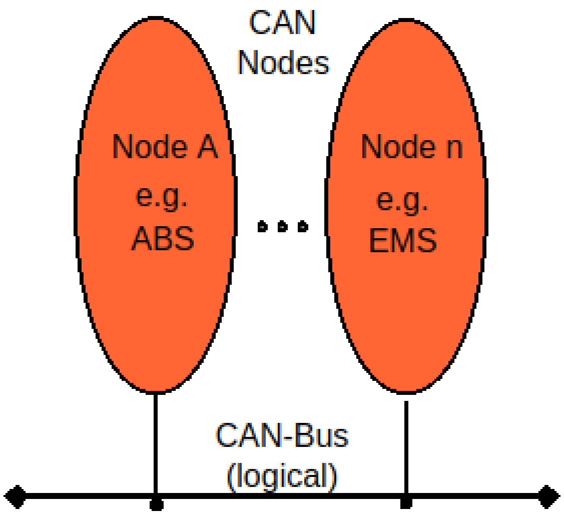


This change results in a more simplistic, cost-effective design. In this example, two CAN standards were used (the High Speed and Low Speed / Fault Tolerant), depending on the designer's preferences on each ECU's maximum data transfer latency.

As a result, different ECUs are able to intercommunicate and perform certain actions, including:

- Auto start/stop: If the car's sensors detect that the vehicle has been stationary for a while, they can send the Engine Control ECU the signal to shut down in order to improve fuel economy and emmisions.
- Air condition: If the driver activates the air contitioner, the dashboard will send a message through the Low Speed CAN to the Air Condition ECU
- Automatic door locks: If the car is stationary for a while or the driver presses a certain button, the dashboard could send a message to the Power Locks ECU in order to lock all the doors. The same components are present in the case of unlocking the doors.

7. Architecture

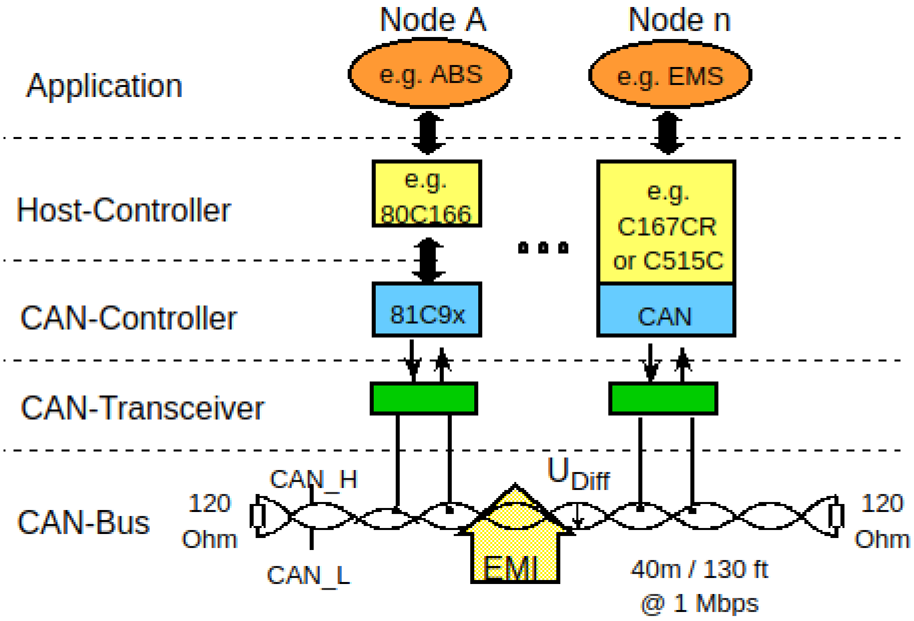


CAN is a multi-master bus with an open, linear structure with one logic bus line and equal nodes. The number of nodes is not limited by the protocol.

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.

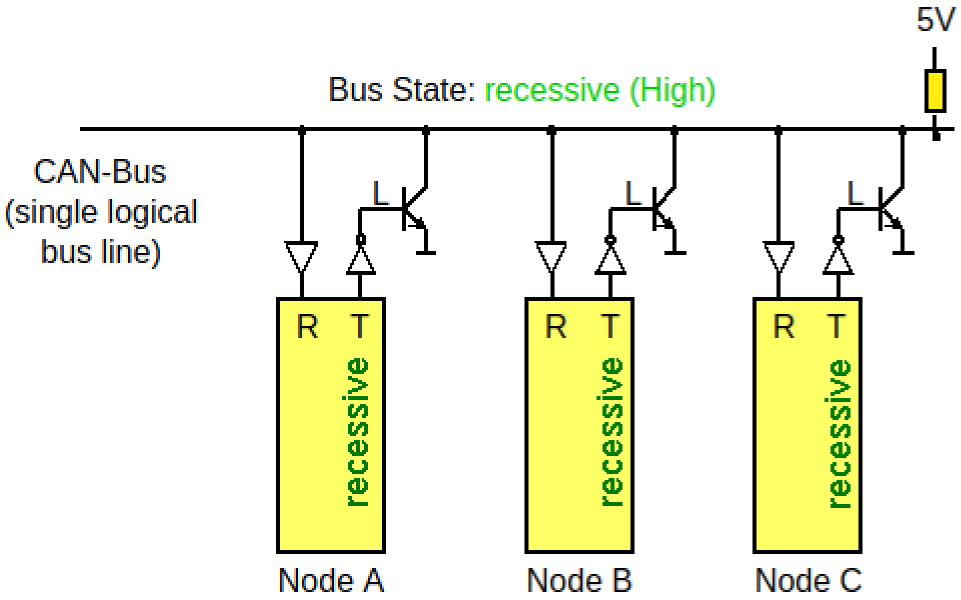
The number of nodes may be changed dynamically without disturbing the communication of the other nodes. Multicasting and Broadcasting is supported by CAN.

8. Data transmission

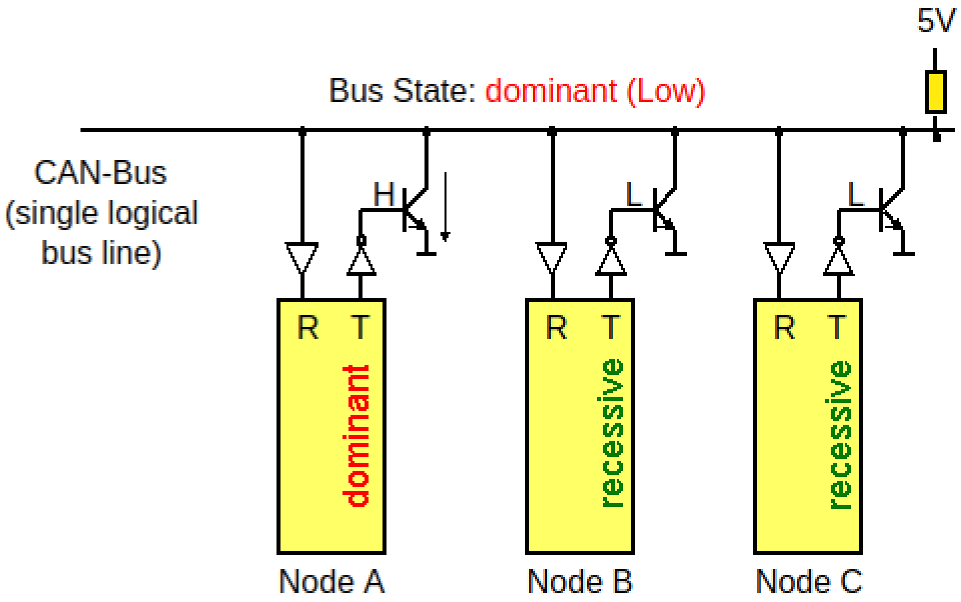


CAN provides sophisticated error-detection and error handling mechanisms such as CRC check, and high immunity against electromagnetic interference. Erroneous messages are automatically retransmitted. Temporary errors are recovered. Permanent errors are followed by automatic switch-off of defective nodes.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

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").

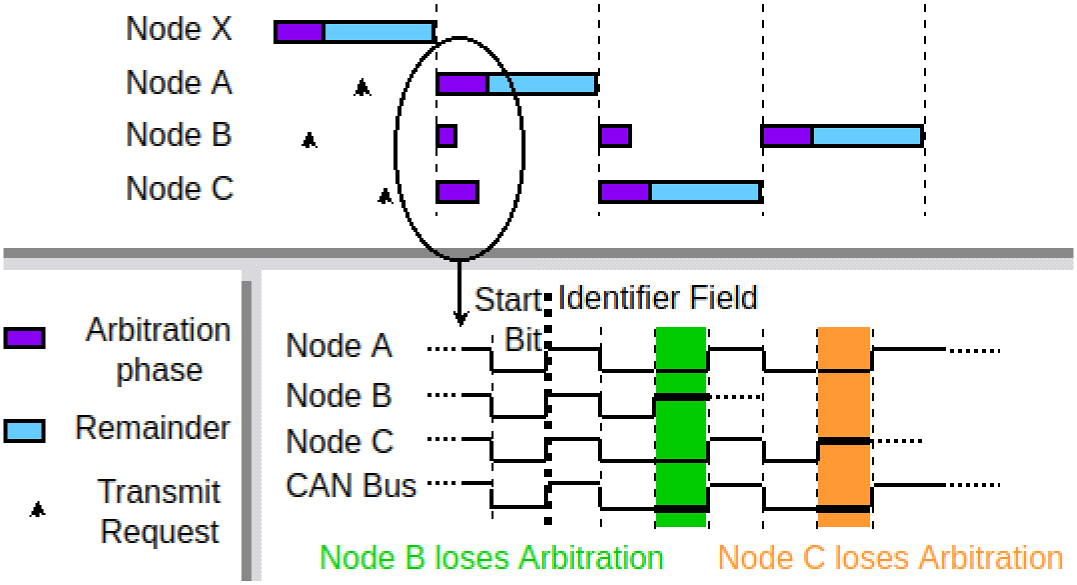


Only if all nodes transmit recessive bits (ones), the Bus is in the recessive state.



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

8.1. Collisions



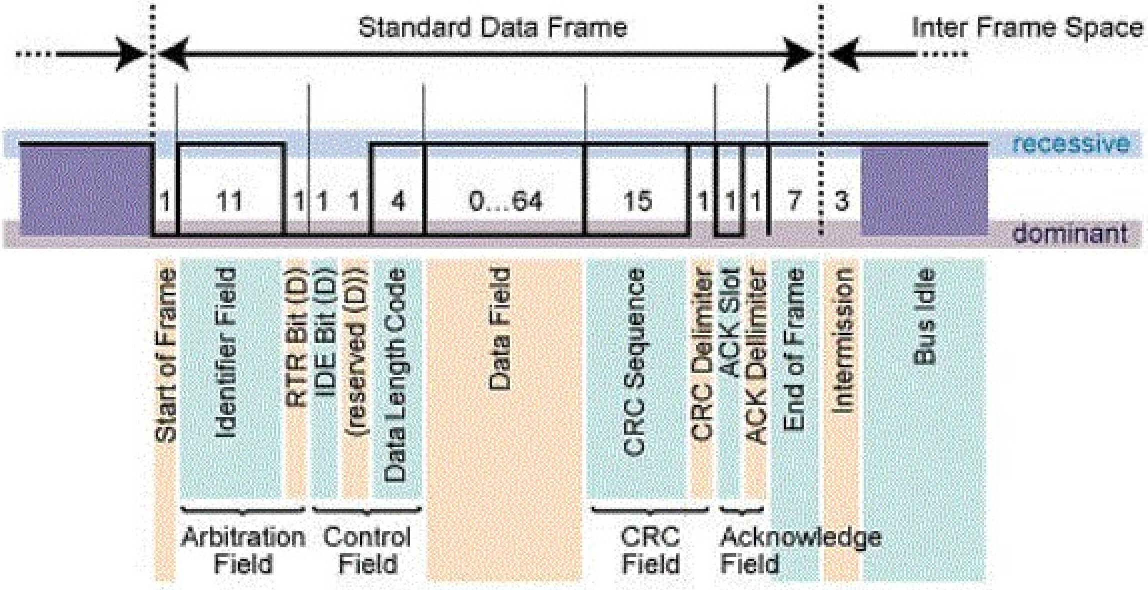
If two or more bus nodes start their transmission at the same time after having found the bus to be idle, collision of the messages is avoided by bitwise arbitration. Each node sends the bits of its message identifier and monitors the bus level.

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.

8.2. Data Frame



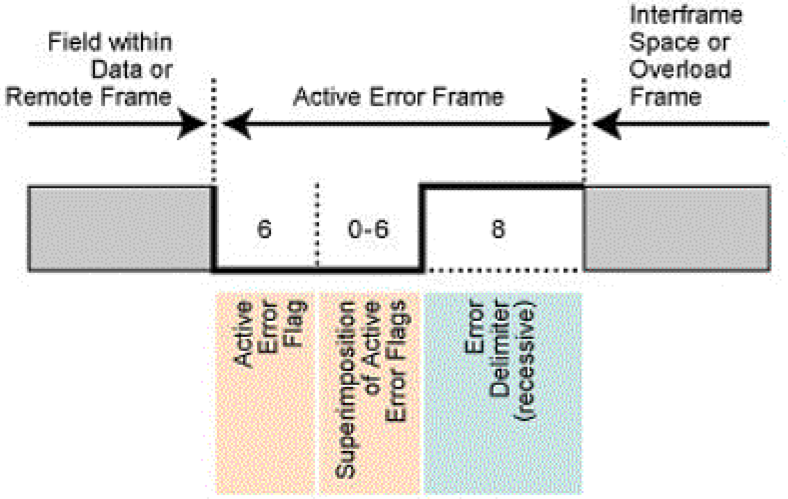
A "Data Frame" is generated by a CAN node when the node wishes to transmit data. The Standard CAN Data Frame is shown above. The frame begins with a dominant Start Of Frame bit for hard synchronization of all nodes.

The Arbitration Field consists 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 Data Length Code (DLC) specifies 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).

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

8.3. Error Frame


Field within Data or Remote Frame
Active Error Frame
Interframe Space or Overload Frame
6
0-6
8
Active Error Flag
Superimposition of Active Error Flags
Error Delimiter (recessive)

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.

8.4. Error checking and handling

The CAN protocol provides sophisticated error detection mechanisms. These mechanisms are:

- 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.
- Acknowledge Check: checks in the Acknowledge Field of a message to determine if the Acknowledge Slot contains a dominant bit.
- Frame Check: If a transmitter detects a dominant bit in the CRC Delimiter, Acknowledge Delimiter, End of Frame or Interframe Space, then a Form Error has occurred.
- Bit Check: if a transmitter either 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, then a Bit Error occurred.
- 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.

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.

9. Bibliography

http://microcontroller.com/learn-embedded/CAN1_sie/CAN1big.htm

http://www.can-wiki.info/doku.php