Controller Area Networks
For Vehicles

by

Hugo Provencher

Directed Studies
ENGR 5004G

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology
April 2012

Abstract

The sophistication and electrification of modern vehicles require more complex control strategies, thus more electronic control units (ECU) interacting with the physical world through actuators and sensors. Therefore, an in-vehicle communication technology connecting these numerous controllers has become essential. This document focuses on a robust in-vehicle communication network used to connect these ECUs: the controller area network communication protocol, also known as CAN bus.

This work is one of the few non-confidential documents available that details CAN bus for vehicles. It covers the fundamentals of the communication protocol, along with hands-on theory and application useful to most CAN users. It provides: an overview on CAN protocol fundamental theory, a description of the hardware required to create a CAN environment, lists of devices supporting CAN available on the market, a rigorous definition of the nomenclature used to accurately define CAN messages and CAN signals, different CAN bus configurations, a variety of techniques to use identifiers, and describes numerous automotive applications.

Keywords: In-vehicle communication, controller area network, CAN bus, ECU, endianness, little-endian, big-endian, network topology, gateway, high-speed CAN, fault tolerant CAN, single-wire CAN, vehicle, CAN transceiver, CAN protocol controller.

Table of Contents

Abstract ii
Table of Contents iii
List of Figures vi
List of Tables vii
List of Abbreviations viii
1. Introduction 1
1.1 Overview 1
1.2 Goal of Manuscript 2
1.3 Summary of Sections 2
2. Properties of CAN Protocol 3
2.1 Basic Concepts 3
2.2 Properties 4
3. Physical Characteristics 5
4. OSI Reference Model 10
4.1 Layered Architecture 10
4.2 Protocols based on CAN 12
4.2.1 Overview 12
4.2.2 CAN for Vehicle Applications 13
5. Data Transmission 14
5.1 Difference between Data Frame and Remote Frame 14
5.2 Start and End of Frames 15
5.2.1 Interframe Space 15
5.2.2 Start of Data and Remote Frames 15
5.2.3 End of Data and Remote Frames 16
5.2.4 Transmission example 16
5.3 Arbitration Field 16
5.3.1 Arbitration Mechanism 17
5.3.2 Frame format: Standard or Extended 18
5.3.3 Frame type: Data or Remote 18
5.3.4 Summary 19
5.4 Control Field 20
5.5 Data Field 20
5.6 CRC Field 21

5.7 ACK Field 21
6. Errors 22
6.1 Type of errors 22
6.1.1 Bit Error 22
6.1.2 Stuff Error 23
6.1.3 CRC Error 23
6.1.4 ACK Error 23
6.1.5 Form Error 23
6.2 Error frame 24
6.3 Overload frame 25
7. CAN Transceiver and Protocol Controller 26
7.1 Overview 26
7.2 CAN Transceiver 27
8. Use of Identifiers 30
9. CAN Message Definition 31
9.1 Message and Signal 31
9.2 Mapping and Positioning of Signals 31
9.2.1 Byte Order (Endianness) 32
9.2.2 Bit Numbering 33
9.2.3 Message Progression 33
9.2.4 Start Bit of Signals 35
9.2.5 Display Formats 35
9.2.6 Examples 36
9.2.6.1 Representation in the 3 main display formats 36
9.2.6.2 Conversion from Intel Standard 37
9.3 Message and Signal Definitions 38
9.3.1 Message Definition 38
9.3.1.1 Filtering 39
9.3.1.2 Post Office Analogy 41
9.3.2 Signal Definition 42
9.3.2.1 Scaling 42
10. Bus Configurations 43
11. Commercial Devices 46
12. Applications for Vehicles 48

12.1 Types of Applications 48
12.1.1 Low Speed Applications 48
12.1.2 High Speed Applications 49
12.1.3 Diagnostic Interface and ECU Programming 50
12.1.4 Gateways between CAN buses 50
12.2 Comparing LIN and CAN 51
13. Conclusion 53
13.1 Summary 53
13.2 Future Development 53
References 55
Appendix A 59

List of Figures

Figure 1. Topology....5
Figure 2. High-speed CAN bus waveform, ISO 11898-2....7
Figure 3. Fault tolerant CAN bus waveform, ISO 11898-3....7
Figure 4. Single wire CAN bus waveform, SAE J2411. 8
Figure 5. DB9 connector....8
Figure 6. 5-pin M12 connector [37]....9
Figure 7. SAE J1962 connector. 9
Figure 8. OSI reference model....10
Figure 9. Physical layer and data link layer detailed [1]....11
Figure 10. CAN based protocols' layers implemented (greyed background). 12
Figure 11. Data frame general structure. 15
Figure 12. Remote frame general structure....15
Figure 13. Interframe Spacing [1]....15
Figure 14. Example of a generic transmission....16
Figure 15. Fastest transmission without overload....16
Figure 16. Slower transmission....16
Figure 17. Arbitration field: Standard format [1]. 16
Figure 18. Arbitration field: Extended format [1]. 16
Figure 19. Arbitration mechanism [15]. 17
Figure 20. Standard data frame [36]. 19
Figure 21. Extended data frame [36]. 19
Figure 22. Standard remote frame [36]....19
Figure 23. Extended remote frame [36]....19
Figure 24. Control field [1]....20
Figure 25. Data field. 21
Figure 26. CRC field [1]. 21
Figure 27. ACK field [1]....21
Figure 28. Error types. 22
Figure 29. Error frame [1]....24
Figure 30. Overload frame [1]. 25
Figure 31. CAN Transceiver, CAN Protocol Controller and Controllers....26
Figure 32. CAN transceiver MCP2551 block diagram [20]....29
Figure 33. Representation of a signal in the 3 main CAN display formats. 36
Figure 34. Conversion from Intel Standard. 37
Figure 35. Bus topology....43
Figure 36. Bus topology minimizing noise....44
Figure 37. Hybrid star-bus topology....44
Figure 38. Daisy chain with short studs....45
Figure 39. Daisy chain with twisted wires in the connector [31]. 45
Figure 40. LIN sub-bus [3]. 52
Figure 41. Safety-critical communication [15]....54
Figure 42. Conversion from Motorola Forward lsb....59
Figure 43. Conversion from Motorola Backward....59

List of Tables

Table 1. Difference between Low Speed and High Speed. 6
Table 2. Bit logic level. 7
Table 3. SAE specifications for CAN buses applications in vehicles [29]. 13
Table 4. Identifier length. 18
Table 5. RTR values. 18
Table 6. DLC encoding method. 20
Table 7. Commercial devices. 27
Table 8. Device and message identifiers. 30
Table 9. Sub-identifiers. 30
Table 10. Message and signal examples. 31
Table 11. Byte numbering and bit order. 32
Table 12. Byte ordering. 32
Table 13. Bit numbering. 33
Table 14. Message progression. 34
Table 15. Display formats. 35
Table 16. Compact form of the 3 main CAN display formats. 36
Table 17. ID filtering example. 40
Table 18. Payload filtering example. 40
Table 19. Post Office Analogy 41
Table 20. Commercial devices. 47
Table 21. Low speed applications. 49
Table 22. High speed applications. 49
Table 23. Comparing LIN and CAN [3]. 51
Table 24. Typical LIN localized support areas. 52

List of Abbreviations

ACK	Acknowledgement
Avg.	Average
Batt.	Battery
BMS	Battery Management System
Byte	1 byte = 8 bits
CAN	Controller Area Network
CANH	CAN High
CANL	CAN Low
CRC	Cyclic Redundancy Check
DLC	Data Length Code
ECU	Electronic Control Unit
EOF	End Of Frame
ID	Identifier
IDE	Identifier Extension
ISO	Independent System Operator
kb/s	kilobits per second
LIN	Local Interconnect Network
LSB	Least Significant Byte
lsb	least significant bit
Mb/s	Megabits per second
MSB	Most Significant Byte
msb	most significant bit
Msg	Message
NRZ	Non Return to Zero
R_S	Slope control
RTR	Remote Transmission Request
RX	Receive
RXD	Receive Digital
SAE	Society of Automotive Engineers
SOF	Start Of Frame
SRR	Substitute Remote Request
Temp.	Temperature
TX	Transmit
TXD	Transmit Digital
UART	Universal Asynchronous Receiver-Transmitter
UOIT	University of Ontario Institute of Technology
V	Volt
V_{DD}	Supply voltage
V_{SS}	Ground
V_{ref}	Reference voltage
Ω	Ohm

1. Introduction

1.1 Overview

The increase in atmospheric pollution partly due to vehicle emissions has caused governments to establish strict laws to sway the automotive industry to build greener automobiles. Vehicle manufacturers are able to satisfy these norms by continuously pushing the level of sophistication in their products. Highly sophisticated vehicles require more complex control strategies, thus more electronic control units (ECU) interacting with the physical world through actuators and sensors. 50 to 100 ECUs can now be found in a single vehicle. Therefore, the information sharing between numerous controllers has become essential. This work focuses on the communication technology connecting those controllers to form a robust network. This efficient and robust field bus is termed controller area network (CAN). Some of the CAN features are its multi-master capability; its built-in error detection and correction capability; as well as its unique fault confinement [1].

Robert Bosch GmbH started developing the controller area network protocol in 1983, but only officially released it at the Detroit's Society of Automotive Engineers (SAE) congress in 1986 [11]. At that time, the automotive industry was looking for a reliable in-vehicle communication system. CAN was developed for use in industrial environments and for in-vehicle networks, but was not the only networking technology competing in the race for the automotive market. Volkswagen Group was developing the A-bus, while another European group was developing the French Vehicle Area Network (VAN). In 1992, Mercedes-Benz built the first car integrating CAN bus and this in-vehicle network then finally won the race in the mid-nineties [11].

Since 1996, the on-board diagnostics II (OBD-II) standard is a mandatory vehicle diagnostics standard [35]. Five communication protocols are included in that standard, CAN being the most important. With the arrival of the CAN-based protocols DeviceNet and CANopen in the mid-nineties, CAN rapidly found applications in factory automation. Since 2000, CAN bus kept expanding to other industries, such as elevators

and forklifts; connections between subsystems in ships, flight status sensors in airplanes and navigation systems in aircrafts; factory and building automation, automatic door control for monorails; different types of industrial controls; medical systems; as well as laboratory and operating room automation.

1.2 Goal of Manuscript

The goal of this work is to gather in a single document a clear, easy and exhaustive description of the controller area network protocol (CAN) for those interested in its fundamental theory, along with relevant and useful details beyond the low level theory directed to a broad range of CAN users.

1.3 Summary of Sections

The basic concepts and properties of the CAN protocol are introduced in Section 2. Section 3 presents its key physical characteristics, such as the basic bus topology, its impedance, ISO standards, bit levels and commonly used connectors. Section 4 introduces the ISO reference model and describes the layers covered by CAN along with CAN-based protocols. In chapter 5, the differences between data frames and remote frames are considered, as well as their fields and the interframe space. In other words, the details of the data transmission are explained. The type of errors, error frames and overload frames are covered in Section 6. Section 7 discusses the roles of the CAN transceiver and protocol controller. A list of commercial devices is also presented in this section. In Section 8, the use of identifiers is explained through examples. The terminology used to discuss CAN messages is detailed in Section 9. Typical bus configurations, such as the bus topology, the star topology and the daisy chain, will be reviewed in Section 10. Section 11 begins with a categorization and descriptions of commercial devices supporting CAN and ends with a list of common ones. The focus of Section 12 is primarily on automotive applications of CAN buses. Future development of CAN technology and a summary can be found in Section 13.

2. Properties of CAN Protocol

2.1 Basic Concepts

CAN bus is a serial communication protocol level supporting real time systems with a high reliability. It handles the detection of collisions, the detection of errors, the retransmission of corrupted messages and the prioritization of sent and received messages. The identifier length can be 11 bits or 29 bits while the data length can vary between 0 and 8 bytes. The bitwise arbitration using the identifier gives a static message priority to the protocol. This property is possible because of the binary logic used by the protocol, either dominant (logical 0) or recessive (logical 1). When a dominant and a recessive bit are simultaneously transmitted, the dominant bit supersedes the recessive one [1].

This protocol uses a producer-consumer multi-master message model, instead of the more common client-server model. Its principal characteristic is the interpretation of the identifier in a message [4]. It does not inform about the destination of the message, but rather indicates where it was sent from; in other words, providing the source of the data.

CAN messages can be transmitted periodically, on request, or on a state change at a uniform and fixed bit rate up to 1 Mb/s, within a CAN bus or CAN network. As explained in the next chapter, the speed, or bit rate, can have different values in different networks. A similar concept applies for the message format: standard or extended.

The CAN communication protocol shows configuration flexibility by requiring no hardware or software modification for any node when a new node is connected to the CAN network. The quantity of nodes on such a network is theoretically unlimited; however, delay times and electrical loads will place an empirical limit on a bus having too many nodes. Sleep mode and wake-up are available options for each node to reduce power consumption [20].

Since the reliability of a communication system is paramount, each node accomplishes powerful safety tasks such as error detection, signalling and self-checking. Each node is able to identify the difference between temporary errors and permanent failures of a node. In the event of a permanent failure, the other nodes on the network automatically command the faulty node to switch off [2].

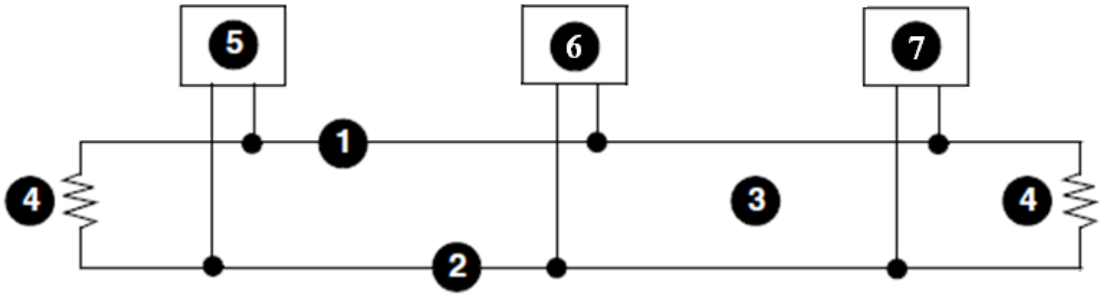
2.2 Properties

In summary, the priorities of the CAN communication protocol are [1]:

- Configuration flexibility
- Prioritization of messages (bitwise arbitration using the identifier)
- Simultaneous reception by multiple nodes with time synchronization
- Multi-master system
- Guarantee of latency time
- Error detection and signalling (by each node)
- Automatic retransmission of corrupted messages (once the bus is idle again)
- Error distinction (between temporary errors and a permanent failure of a node)
- Fault confinement (automatic switch off of defective nodes)

3. Physical Characteristics

The CAN specification does not define how the single channel which carries bits is implemented. A common way to implement a CAN bus is by using a single wire and a ground. However, the most typical implementation and the main focus of this paper is using 2 twisted differential wires, CAN high and CAN low, with 2 termination resistors of 120 ohms each. Figure 1 shows a typical CAN bus topology when two differential wires are used, based on reference [4].



#	Description
1	CAN High wire (CANH)
2	CAN Low wire (CANL)
3	Voltage between CANH and CANL
4	Termination resistors (120Ω)
5	Node (i.e. ECU #1)
6	Node (i.e. ECU #2)
7	Node (i.e. ECU #3)

Figure 1. Topology

Depending on the configuration of the network and the environment, the transmission distance can reach up to 1 km. Table 1 summarizes 3 common specifications of the protocol.

Table 1. Difference between Low Speed and High Speed.

Parameters	ISO 11898-2	ISO 11898-3	SAE J2411
Name	high-speed	fault tolerant (low-speed)	single-wire
Baud rate	up to 1Mb/s	up to 125kb/s	33.3kb/s (up to 83.3kb/s)
Number of wires	2	2	1
Dominant bit level	CAN-H = 3.5V CAN-L = 1.5V	CAN-H = 4V CAN-L = 1V	3.6V
Recessive bit level	CAN-H = 2.5V CAN-L = 2.5V	CAN-H = 1.75V CAN-L = 3.25V	0V
Termination	2 resistors of 120Ω	Each node needs to terminate both CAN lines individually	9,09kΩ load resistor
Length	Limited by busload and data rate 40m @ 1Mb/s 100m @ 250 kb/s 1km @ 50kb/s	Limited by busload and data rate	
Number of nodes	Limited by busload and data rate	up to 32	up to 32
Maximum bandwidth	2000 msg/s @ 250 kb/s		
	4000 msg/s @ 500 kb/s		
	8000 msg/s @ 1 Mb/s		

As it will be explained in Section 7.2, the bus load associated with CAN transceivers, resistors and capacitors, whereas the maximum speed and length are dependent on the acknowledgment bit. The field of applications of each of these 3 specifications is broad. Section 12.1 details applications in the automotive industry. To assure robust arbitration when designing a network, the bandwidth should ideally be kept below 70% of the maximum bandwidth [5]. Bits travel on the bus using a non-return to zero (NRZ) bit encoding and decoding. This means the bit level is maintained over a full bit time and changes at the following bit time if a complementary bit is transmitted [10]. For synchronization purposes, a maximum of 5 consecutive bits of equal logic level is allowed before the insertion of a complementary bit [1]. This technique is called bit stuffing. As explained previously, the 2 bit levels are defined as dominant and recessive, and are associated with the logic level in Table 2.

Table 2. Bit logic level.

Bit	Logic Level
Dominant	0
Recessive	1

Below is an example of a CAN high-speed transmission showing physical bits on a CAN bus, and therefore the NRZ coding method:

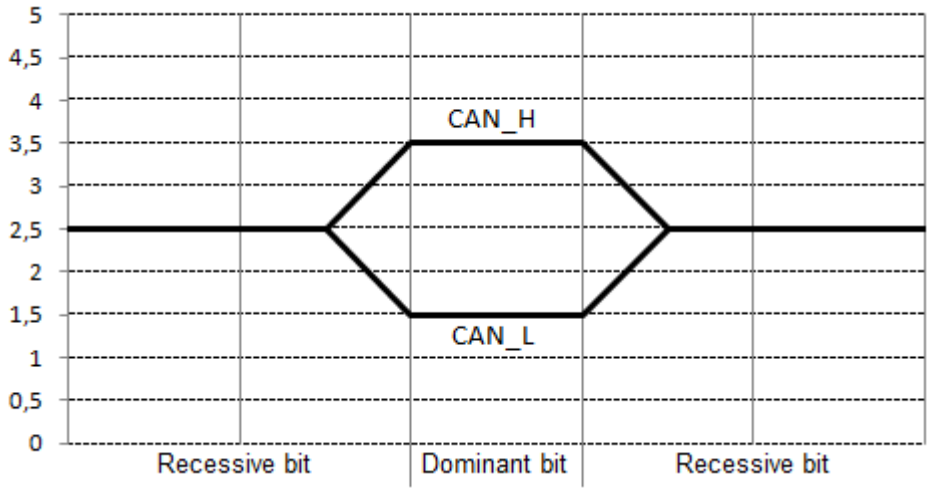


Figure 2. High-speed CAN bus waveform, ISO 11898-2.

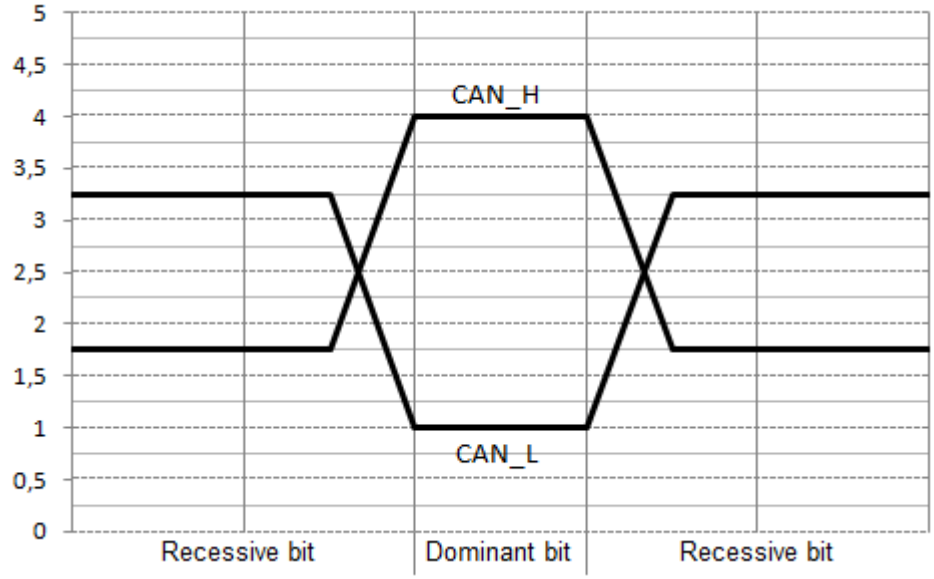


Figure 3. Fault tolerant CAN bus waveform, ISO 11898-3.

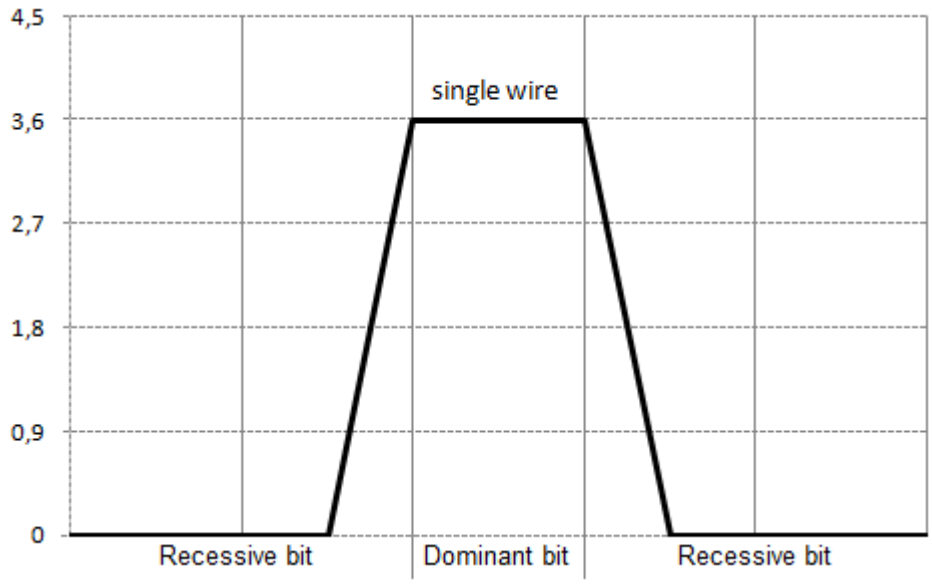
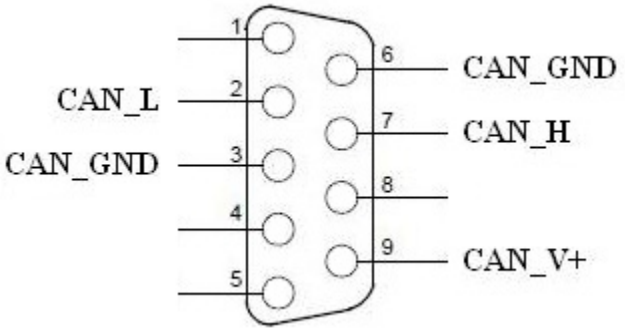


Figure 4. Single wire CAN bus waveform, SAE J2411.

There are 2 standard types of connectors to access a CAN bus: DB9 and 5-pin M12, respectively Figure 5 and Figure 6.



Pin	Signals	Description
1		
2	CAN_L	Low dominant level
3	CAN_GND	Ground
4		
5		
6	CAN_GND	Ground(optional)
7	CAN_H	High dominant level
8		
9	CAN_V+	Power (optional)

Figure 5. DB9 connector.

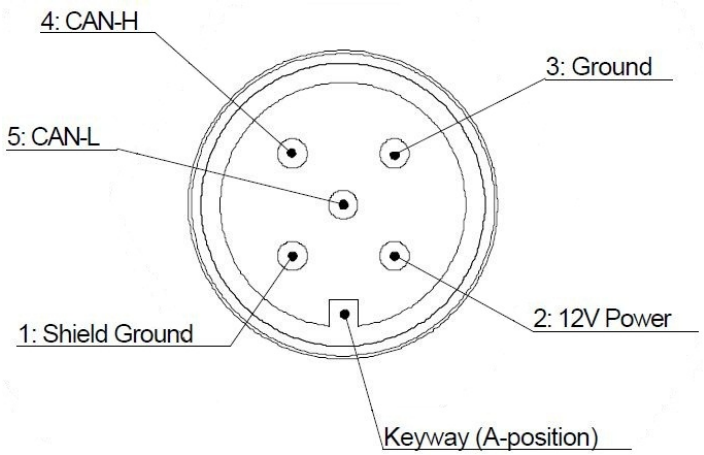
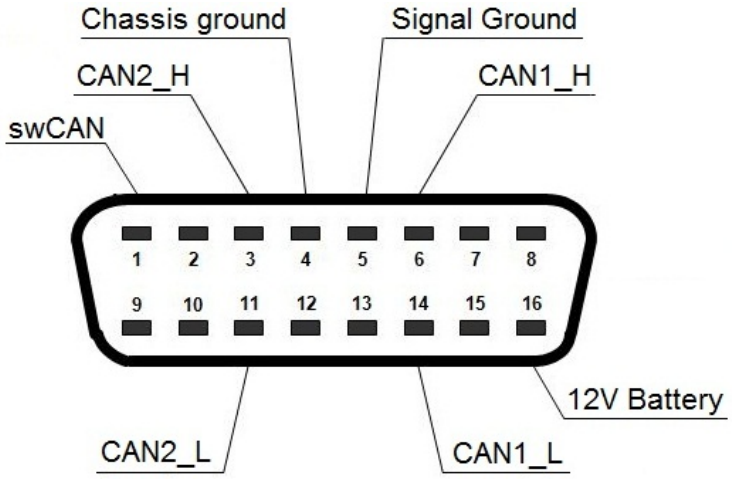


Figure 6. 5-pin M12 connector [37].

It is also interesting to note that in the automotive industry the connector used to access the CAN buses on a vehicle is the SAE J1962 (on-board diagnostics OBDII) connector as shown in Figure 7, based on reference [29].



Pin #	Description
1	Single wire CAN
2	
3	CAN2 High
4	Chassis ground
5	Signal ground
6	CAN1 High
7	
8	
9	
10	
11	CAN2 Low
12	
13	reserved
14	CAN1 Low
15	
16	Battery +12V

Figure 7. SAE J1962 connector.

4. OSI Reference Model

4.1 Layered Architecture

Every communication protocol can be represented by an open system interconnection reference model (OSI model) having 7 layers: application, presentation, session, transport, network, data link and physical [24]. This model divides the characteristics into layers within the communication protocol. Not every communication specification details each of the 7 layers. For example, CAN specifies only the 2 basic layers, the physical layer and the data link layer as illustrated by the 2 shaded boxes on Figure 8.

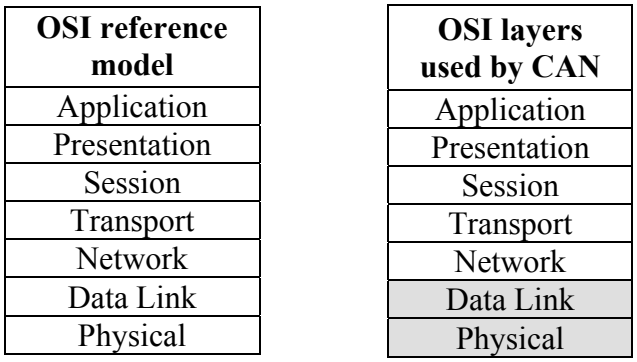
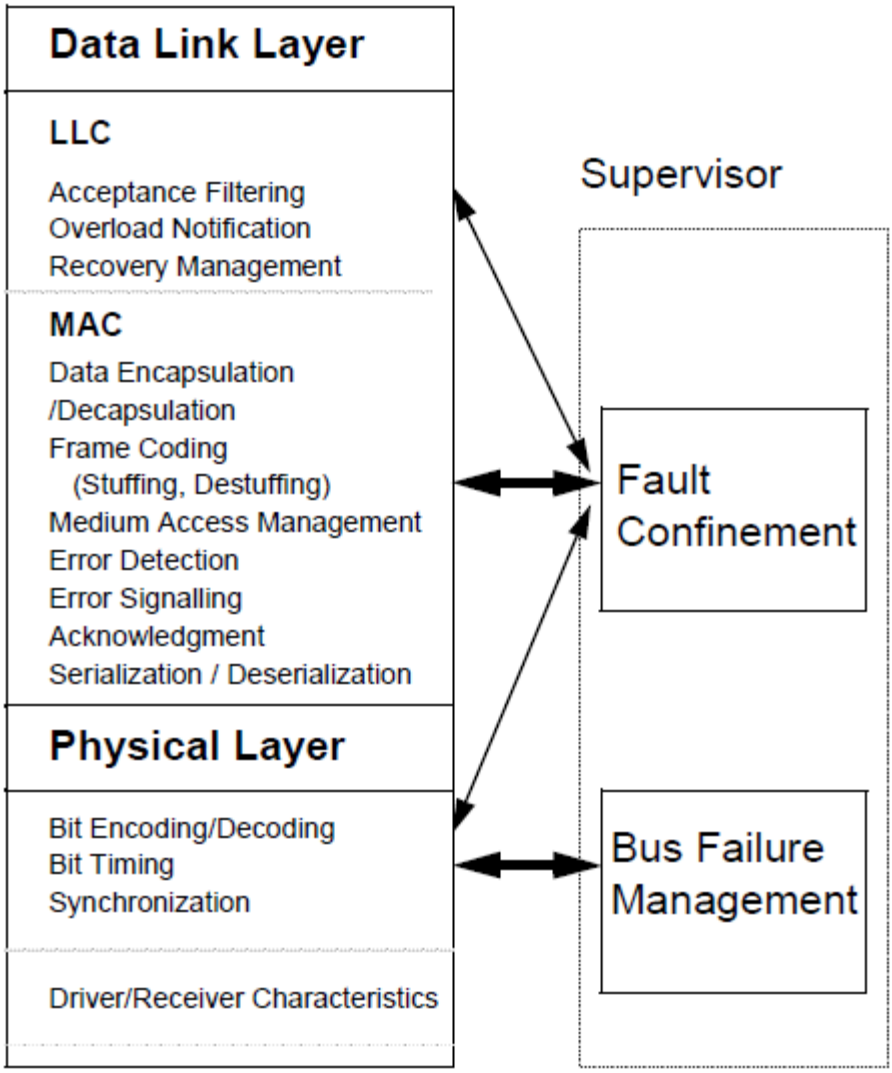


Figure 8. OSI reference model.

The physical layer defines the physical transmission of a frame between 2 nodes. It also standardizes the bus electrical characteristics, i.e. the dominant and recessive bit voltages, and its mechanical characteristics, such as the connectors. As it will be explained in Section 7, CAN transceivers are associated with this layer.

The data link layer transforms the physical layer into a data link free of transmission errors. It divides the data sent and received into frames. In summary, it defines the role of the CAN protocol controller as detailed in Section 7. Figure 9 details the physical and the data link layers of the CAN communication protocol.



LLC: Logical Link Control

MAC: Medium Access Control

Figure 9. Physical layer and data link layer detailed [1].

4.2 Protocols based on CAN

4.2.1 Overview

Important CAN based protocols [9][29]:

- CAL (CAN Application Layer) (automation), by CiA in 1992
- DeviceNet (factory and process automation), by Allen-Bradley in 1994
- CANopen (embedded network in machine controls), by CiA in 1995
- TTCAN (Time-Triggered CAN), created in 2000
- SAE J2284 (vehicles), by SAE in 2002
- SAE J2411 (vehicles), by SAE in 2002
- SAE J1939 (heavy duty commercial trucks, buses)
- J1939-based

On top of the physical and data link layer general definitions, most of these CAN based protocols, CAL, CANopen and DeviceNet, specifies how to interface data from the network to the user, known as the application layer . It gives the user access to the basic functionalities of the CAN network [10]. Others, such as SAE J2284 and SAE J2411, only define the physical layer and portions of the data link one [29]. Figure 10 summarizes the implemented OSI layers (greyed) of some common CAN based protocols.

CAN	**CAL**	**CANopen**	**DeviceNet**	**SAE J2284 SAE J2411**
Application	Application	Application	Application	Application
Presentation	Presentation	Presentation	Presentation	Presentation
Session	Session	Session	Session	Session
Transport	Transport	Transport	Transport	Transport
Network	Network	Network	Network	Network
Data Link	Data Link	Data Link	Data Link	Data Link
Physical	Physical	Physical	Physical	Physical

Figure 10. CAN based protocols' layers implemented (greyed background).

The application layer defines a way to interface data from the network to the user and is the only extra layer implemented in several CAN based protocols. It gives the user access to the basic functionalities of the CAN network.

4.2.2 CAN for Vehicle Applications

Table 3 groups the 3 different subdivisions of the high-speed CAN specification for vehicle applications SAE J2284, and the specification of the single-wire version SAE J2411 [29]. As explained in the previous section, these specifications are based on the 2 OSI layers defined by CAN. However, they define them more precisely, i.e. with more restrictions. For example, these SAE specifications specify to use non-shielded wires.

Table 3. SAE specifications for CAN buses applications in vehicles [29].

Name	Description	Status
SAE J2284-1	High-speed CAN for vehicle applications @ 125kb/s	Issued: 07-Mar-2002
SAE J2284-2	High-speed CAN for vehicle applications @ 250kb/s	Issued: 07-Mar-2002
SAE J2284-3	High-speed CAN for vehicle applications @ 500kb/s	Revised: 02-Mar-2010
SAE J2411	Single-wire CAN for vehicle applications	Issued: 14-Feb-2000

SAE J2284-3, providing a higher baud rate, and SAE J2411, presenting a cheaper solution, seem to be more common in vehicles then SAE J2284-1 and SAE J2284-2. Being the only CAN specification updated and re-published by the Society of Automotive Engineers since their first release, it is safe to assume that SAE J2284-3 will continue to be used for automotive applications in the foreseeable future.

5. Data Transmission

There are four different frame types [1]:

- Data Frame (standard or extended)
- Remote Frame (standard or extended)
- Error Frame (generated by the protocol manager)
- Overload Frame (generated by the protocol manager)

This section will focus on the data frame, but also cover the remote frame since it is a particular case of the data frame. The next section will explain error and overload frames.

The data frame can be divided in 7 subsections [1]:

- Start of Frame
- Arbitration Field
- Control Field
- Data Field
- CRC Field
- ACK Field
- End of Frame

5.1 Difference between Data Frame and Remote Frame

The remote frame is a data frame having a payload equal to 0, it carries no data. It is used to request data from specific nodes. Another difference between these 2 frames is the value of the remote transmission request (RTR) bit in the arbitration field. Figure 11 and Figure 12 detail the general structure of data frames and remote frames, translated from [4].

Start of Frame (1bit)	Arbitration field (12bits or 32bits)	Control field (6bits)	Data field (1 to 8 bytes)	CRC field (16bits)	Ack field (2bits)	End of Frame (7bits)

Figure 11. Data frame general structure.

Start of Frame (1bit)	Arbitration field (12bits or 32bits)	Control field (6bits)	CRC field (16bits)	Ack field (2bits)	End of Frame (7bits)

Figure 12. Remote frame general structure.

5.2 Start and End of Frames

5.2.1 Interframe Space

The interframe space field has a minimum length of 3 recessive bits, called the intermission, preceded by data frames and remote frames, but not overload or error frames. The interframe space can be longer if the bus is idle or if an error occurs, since the transmission will be suspended for 8 recessive bits. The bus is idle when no node is transmitting, and therefore has an arbitrary length.

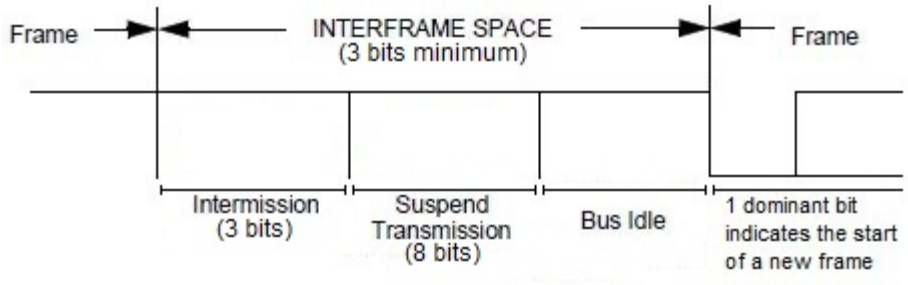


Figure 13. Interframe Spacing [1].

5.2.2 Start of Data and Remote Frames

Data and remote frames begin with a start of frame bit (SOF). One dominant bit during the interframe space indicates the start bit of a new frame.

5.2.3 End of Data and Remote Frames

Both frames always end with 7 recessive bits, called the end of frame (EOF), and are followed by an interframe space [1]. In other words, the differential voltage between CAN high and CAN low is 0V.

5.2.4 Transmission example

An example of a generic transmission is shown on Figure 14.

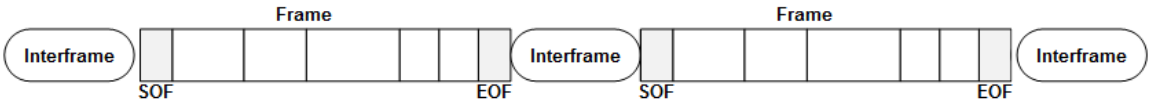


Figure 14. Example of a generic transmission.

Figure 15. Fastest transmission without overload.

Figure 16. Slower transmission.

5.3 Arbitration Field

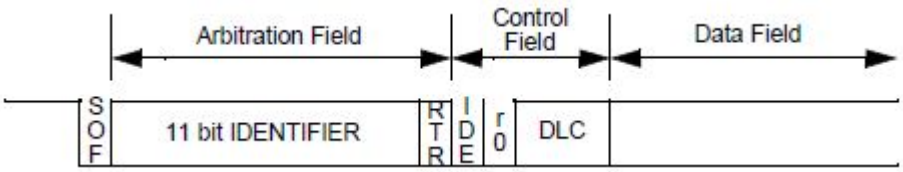


Figure 17. Arbitration field: Standard format [1].

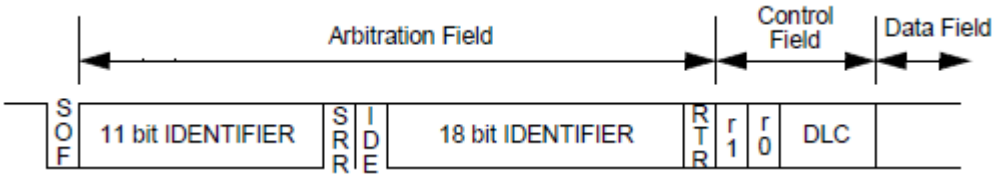


Figure 18. Arbitration field: Extended format [1].

5.3.1 Arbitration Mechanism

As explained in Section 2.1, when a dominant bit (0) and a recessive bit (1) are simultaneously transmitted, the dominant bit overrides the recessive one. This is the fundamental principle under the bitwise arbitration mechanism using identifiers. Figure 19 shows an example of arbitration by using 3 nodes that attempt to send a message simultaneously and gain access to the CAN bus.

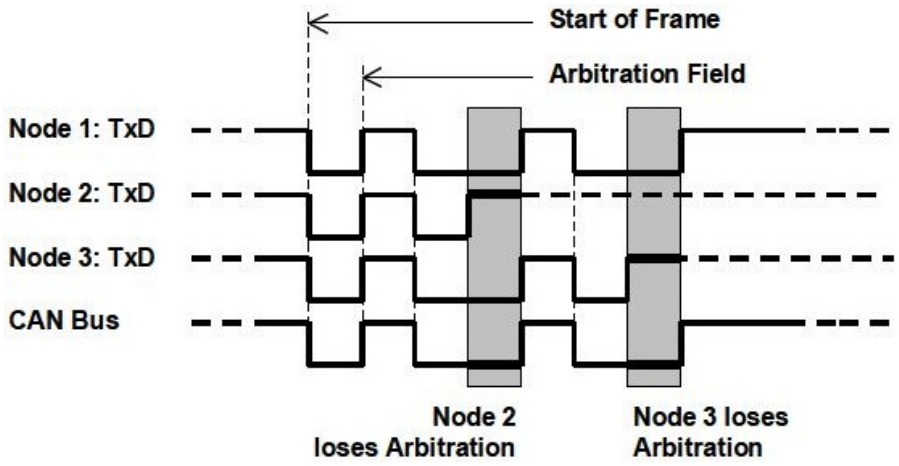


Figure 19. Arbitration mechanism [15].

The grey area on the left overlaps the 3rd bit of the arbitration field of each signal. Node 1 and 3 have a dominant bit (0) while node 2 is attempting to send a recessive one (1). Since the 3rd bit of node 2's identifier is overridden, node 2 realizes that one or several messages of higher priority are trying to access the bus. Thus, it will stop transmitting and re-attempt to send its message as soon as the bus is free again.

A similar scenario happens to node 3 on the 6th bit of the identifier shadowed by the right grey area. Trying to send a recessive bit while node 1 sends a dominant one, node 3 ceases sending its message. Node 1 wins the arbitration, thus gets access to the bus and finishes transmitting its message. As detailed under Section 7.2, by sending a dominant bit, physically node 1 is pulling-up CANH and pulling-down CANL, while node 3 is trying to maintain both CANH and CANL at 2.5V and loses arbitration. Node 3 detects the voltage difference on the line, then gives up and relinquished till the bus is idle again.

Whenever the bit a node is attempting to transmit during the arbitration field is different than the physical result on the CAN bus, which is illustrated as the bottom signal on Figure 19, the node in question stops transmitting and waits the end of frame before re-attempting to send it once more. From another perspective, the node winning the arbitration is identical to the result on the CAN bus [15].

5.3.2 Frame format: Standard or Extended

The main subdivision in the arbitration field is the identifier. The CAN specification 2.0 offers 2 possibilities regarding the length of the identifier, standard (11 bits) or extended (29 bits) [1]. These are referred to as the frame format. The substitute remote request (SRR) bit is a recessive bit in the remote transmission request (RTR) position of the extended format. In the extended format, the identifier extension (IDE) bit is recessive and part of the arbitration field while it is dominant and part of the control field in the standard format.

Table 4. Identifier length.

Frame Format	Identifier Length
Standard	11 bits
Extended	29 bits

5.3.3 Frame type: Data or Remote

As explained previously, a device can request another device to send back a specific data frame by sending a remote frame. Such a frame contains no data and is recognized by the RTR bit located at the end of the identifier in the arbitration field [1].

Table 5. RTR values.

Frame Type	RTR	Logic Level
Data	Dominant	0
Remote	Recessive	1

5.3.4 Summary

This section depicts in detail the 4 possible combinations of frames discussed above.

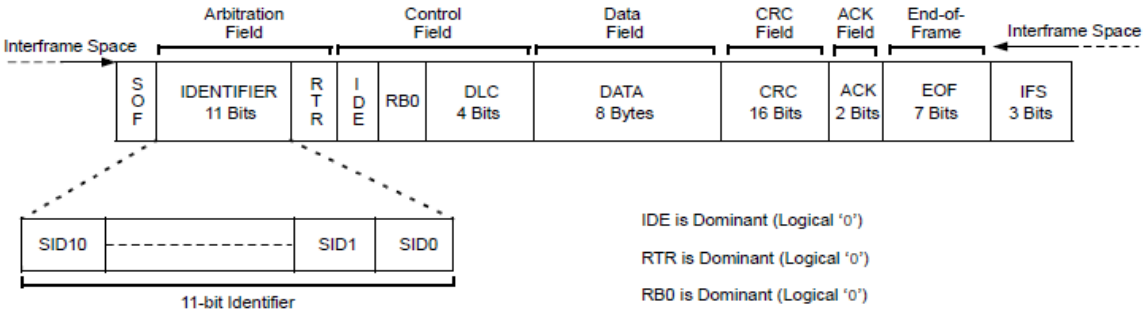


Figure 20. Standard data frame [36].

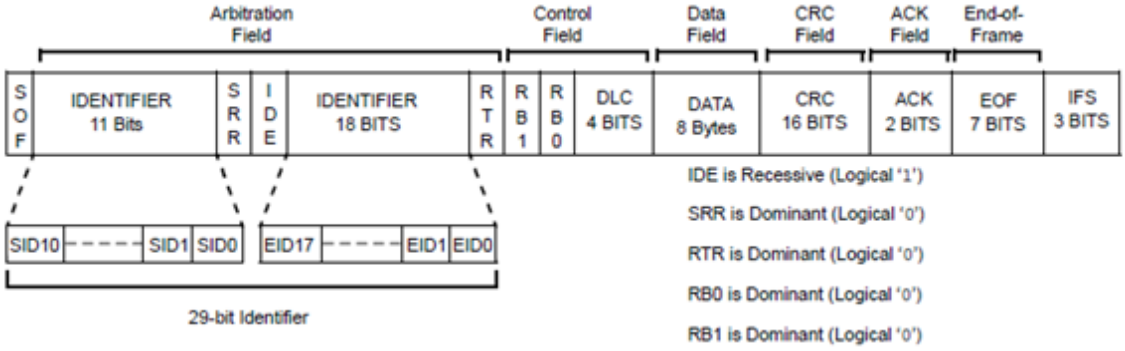


Figure 21. Extended data frame [36].

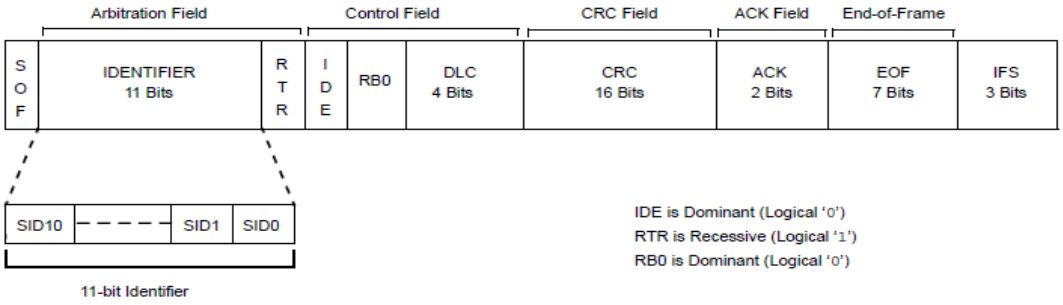


Figure 22. Standard remote frame [36].

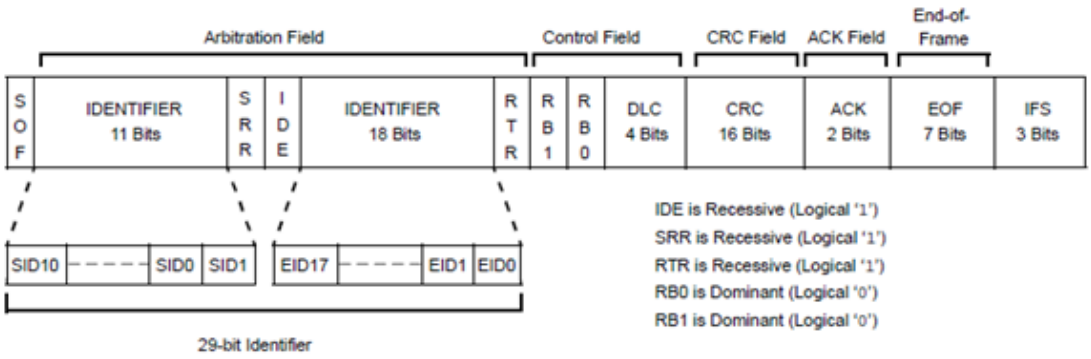


Figure 23. Extended remote frame [36].

5.4 Control Field

The control field (6 bits) is divided in 2 subsections, reserved bits (2 bits) and data length code (DLC) (4 bits). The r1 and r0 bits are reserved to ensure future compatibilities for frames in the extended format. In the standard format, the identifier extension (IDE) is a dominant bit and only r0 is reserved. The DLC3 to DLC0 bits define how many bytes of data will follow in the data field. Figure 24 represents the control field and Table 6 shows the DLC encoding method, data extracted from reference [1].

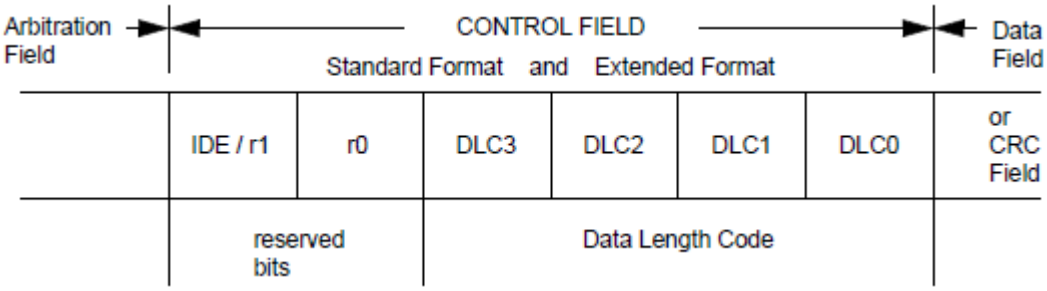


Figure 24. Control field [1].

Table 6. DLC encoding method.

Data field length (bytes)	Frame type	DLC			
		DLC3	DLC2	DLC1	DLC0
0	Remote	D	D	D	D
1	Data	D	D	D	R
2	Data	D	D	R	D
3	Data	D	D	R	R
4	Data	D	R	D	D
5	Data	D	R	D	R
6	Data	D	R	R	D
7	Data	D	R	R	R
8	Data	R	D	D	D

D : Dominant bit (bit = 0), R : Recessive bit (bit = 1)

5.5 Data Field

The frame format has no effect on the data field, only the frame type and DLC influence this field. For data frames, the length of the data field may vary from 1 to 8 bytes and is defined by the DLC [1]. However, the payload, the size of the data field, is always 0 bytes for remote frames.

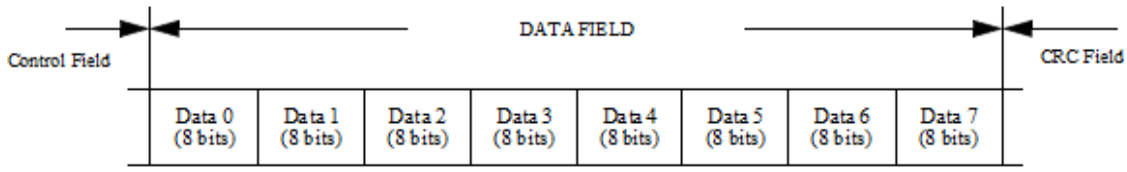


Figure 25. Data field.

5.6 CRC Field

CRC stands for cyclic redundancy check. It checks the validity of the SOF, arbitration field, control field and data field. It can detect a maximum of 5 errors in the message. This field ends with a recessive bit delimitation called the CRC delimiter [1].

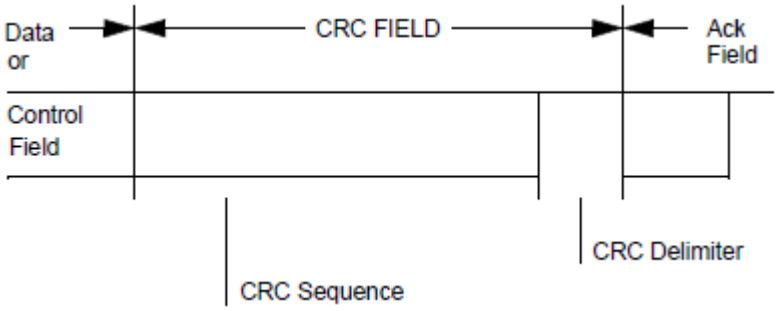


Figure 26. CRC field [1].

This field is described further under Section 6.1.3.

5.7 ACK Field

The first bit of the acknowledgement field is dominant if no error related to the CRC has been detected by other nodes on the bus. Otherwise, an error frame is sent. The second bit is always recessive and is the acknowledgment delimiter [1]. This bit is followed by a flag sequence of seven recessive bits known as the end of frame as explained in Section 5.2.3.

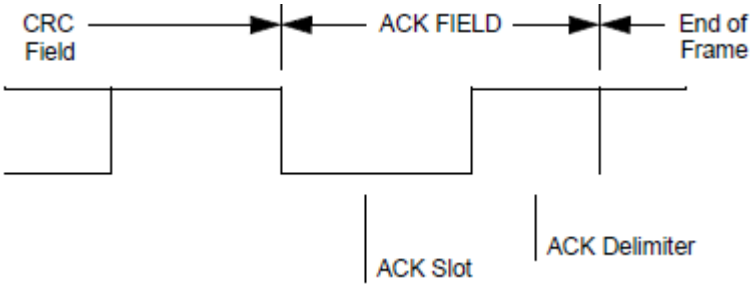


Figure 27. ACK field [1].

6. Errors

6.1 Type of errors

Errors are managed by the two subsequent frames: error frames and overload frames [1]. An error frame is sent by all the nodes detecting an error. To ask for more time following data frames and remote frames, an overload frame is transmitted. The following 5 types of errors exist in CAN communication and are displayed on Figure 28, adapted from reference [2]:

- Bit error (monitoring error)
- Stuff error
- CRC error (and CRC delimiter error)
- ACK error (and ACK delimiter error
- Form error (message frame check)

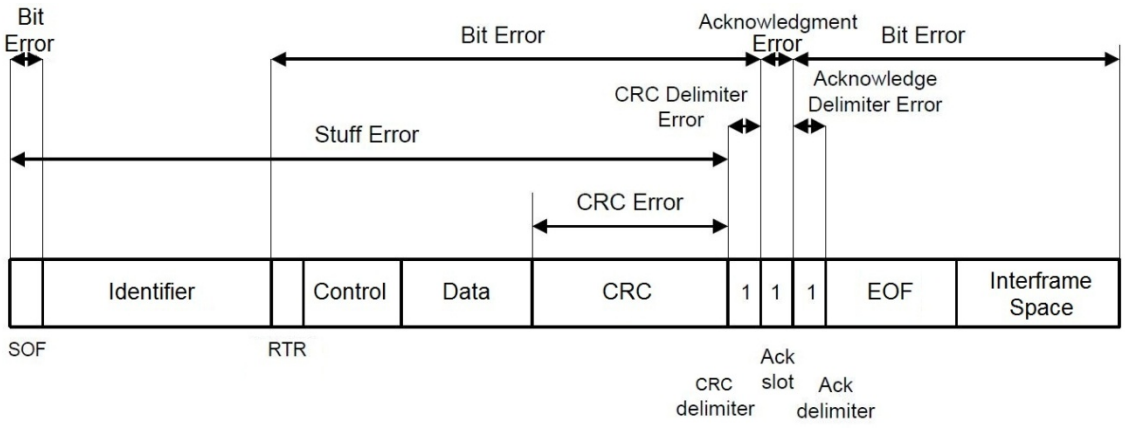


Figure 28. Error types.

6.1.1 Bit Error

A bit error is detected by the unit sending the frame itself. When sending data on the bus, the unit monitors the bus to check if the bit sent is at the desired logic level. If not, an error frame is sent. However, 3 exceptions exist. There is no bit error verification during the transmission of arbitration fields, acknowledgement slots and passive error flags [2]. Passive error flags are discussed in the next section.

6.1.2 Stuff Error

The protocol does not allow the transmission of more than 5 consecutive equal bit levels. Therefore, a stuff error occurs when 6 or more consecutive equal bit values are detected. When 5 consecutive bits of the same logic level are sent, a stuffing bit of the opposite logic level should be stuffed after the 5^{th} consecutive bit in the message. This procedure is called coding by the method of bit stuffing [2].

6.1.3 CRC Error

The transmitter calculates the cyclic redundancy check (CRC) value according to the bits sent in the SOF, arbitration field, control field and data field. This value is calculated again by the receivers. If the CRC value of the transmitter differs from that of the receivers', a CRC error is detected and an error frame is sent [2].

6.1.4 ACK Error

As explained previously, the transmitter sends within the acknowledgement field a recessive bit at the ACK slot. This bit has to be overwritten by a dominant one sent by one of the receivers. If the transmitter does not monitor a dominant bit in the ACK Slot, it detects an ACK error [2].

6.1.5 Form Error

In a frame, some bits have a fixed-form, such as the CRC delimiter, the ACK delimiter and the EOF field. If one or more of these bits are illegal, a form error is detected [2].

6.2 Error frame

An error frame has the two following fields: error flag and error delimiter [1]. The active error flag (6 dominant bits) and the passive error flag (6 recessive bits) are the 2 existing flags used [2]. It should be noted that the bit-stuffing rule does not apply to these flags. The way to recover from an error is by automatically re-transmitting the faulty message from the transmitter. Each node has an error counter determining whether it is in active error mode, passive error mode or bus off mode. When a station is in active error mode, it sends an active flag on error detection. The opposite applies for one in passive error mode. In other words, by sending flags with dominant bits, nodes in active error mode have priority over the ones in passive error mode. The general structure of an error frame is illustrated in Figure 29.

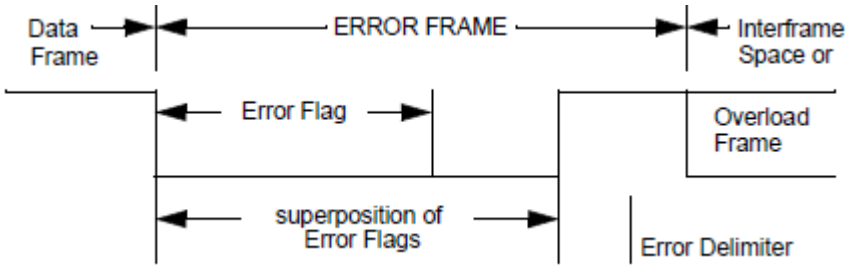


Figure 29. Error frame [1].

The station, or node, sending the first error flag sends 6 dominant bits when in active mode, breaking the bit stuffing rule. When the bit-stuffing error is detected, the other nodes, in either active or passive modes, start sending active and passive flags respectively. In this case, passive error flags are ignored, because they are superscribed by the active flags' dominant bits. As a consequence, the value of the bus becomes a superposition of error flags varying from 6 to 12 dominant bits (see Figure 29). This superposition does not affect the length of each nodes' error flag, but does affect the overall error active flag on the bus which can be of a maximum length of 2 active error flags. Using recessive bits, a passive error flag is much simpler and is always 6 recessive bits long regardless of the superposition of other passive error flags from other nodes that might overlap with the error delimiter. However, it can be overwritten by an active error flag [2].

The error delimiter is 8 recessive bits following an active error flag or a passive error flag. In the first case, there is a superposition of passive error flags and error delimiters that are overwritten by active error flags. Once the last active error flag is completely sent, after 6 to 12 dominant bits, the first recessive bit of the overall error delimiter appears. In the second case, no node is sending dominant bits resulting in the start of the overall error delimiter after exactly 6 recessive bits during the superposition of passive error flags [2].

6.3 Overload frame

There are 2 causes resulting in the generation of an overload frame [2]:

- A receiver requiring a delay before being able to decode the next data frame or remote frame.
- A node detecting a dominant bit in the first and second bit positions during the intermission field of the interframe space.

In both cases, an overload frame, represented in Figure 30, has the general form of an error frame containing active error flags. In other words, 6 to 12 dominant bits as superposition of overload flags and 8 recessive bits as the overload delimiter are sent.

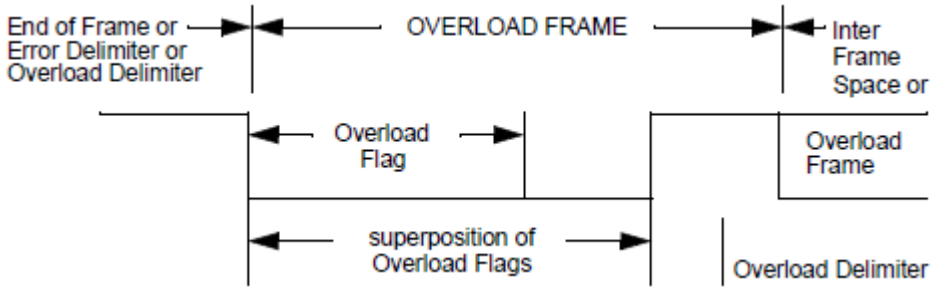


Figure 30. Overload frame [1].

7. CAN Transceiver and Protocol Controller

7.1 Overview

It is not really necessary to know all the details in the transmission management of error frames, overload frames, the SOF, the ACK and the EOF. Most CAN users, for instance application developers, only need to understand the general idea of the arbitration field (standard identifier or extended identifier), the control field (encoding method of the DLC) and the data field (endianness) [5].

The CAN protocol controller takes care of the details, such as applying the protocol specifications. It automatically handles the protocol and errors. However, a CAN transceiver is required to convert the bus voltage to a logic level value and to carry out the opposite conversion. Even though most microcontrollers supporting CAN have an integrated CAN protocol controller improving performance, they still need CAN transceivers, sometime called line drivers. Nevertheless, commercial controllers have these two devices integrated into their circuitry. Figure 31 summarizes these 3 cases and Table 7 gives examples of different products on the market.

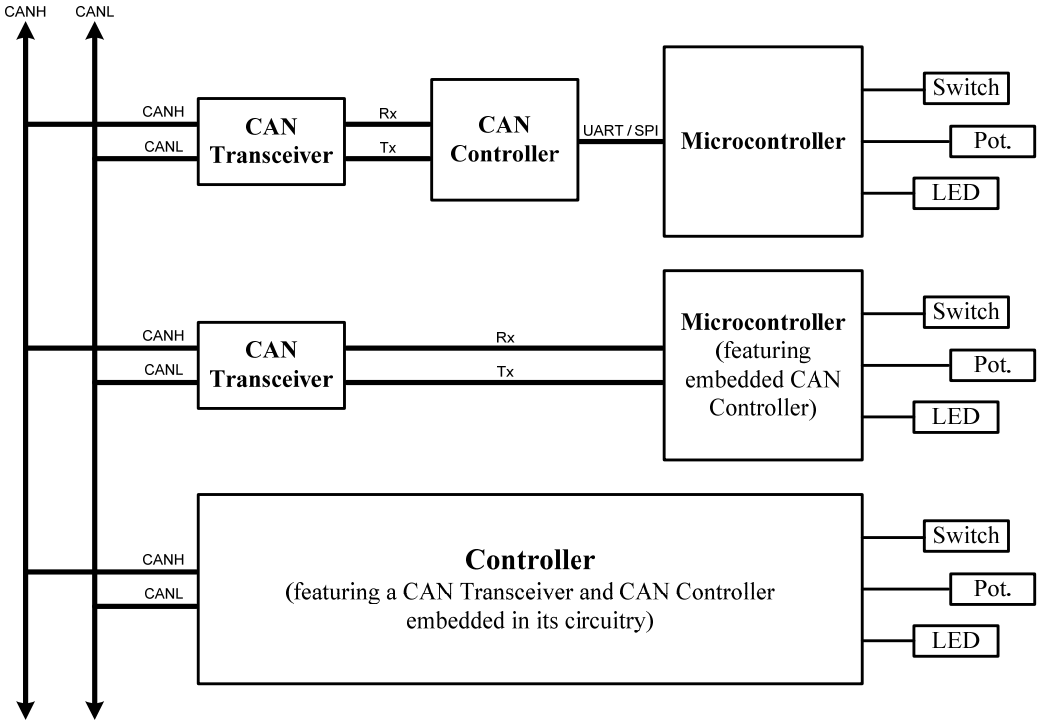


Figure 31. CAN Transceiver, CAN Protocol Controller and Controllers.

Table 7. Commercial devices.

CAN Transceivers	CAN Protocol Controllers	Microcontrollers with CAN	Commercial Controllers
High-Speed	**High-Speed**	**Microchip**	**MotoHawk**
MCP2551 (HS)	MCP2515 (HS)	dsPIC33FJ256GP710	ECM-5554-112
TJA1041 (HS)	SJA1000 (HS)	PIC24HJ256GP610A	ECM-0555-080
TJA1050 (HS)		**Atmel**	ECM-0565-128
MAX3050 (HS)		AT90CAN128	**National Instruments**
MAX3057 (HS)		AT89C51CC03	PCI/PXI/PCMCIA
ATA6660 (HS)		**Freescale Semiconductor**	**Vector**
Fault Tolerant		MPC5554	CANlog4
TJA1054 (LS)		MPC555	CAN Case XL
MC33388 (LS)		MPC565	**Others**
Single Wire			CAN-AC2-PCI
TLE6255G (SW)			CANview USB

7.2 CAN Transceiver

Figure 32 represents the block diagram of a basic high-speed CAN transceiver, the MCP2551 manufactured by Microchip [20]. It is equivalent to the TJA1050 made by NXP Semiconductors, known as Philips Semiconductors before 2006 [22]. The TJA1041 is more sophisticated and will not be discussed in this paper, although its basic functionalities are similar to the MCP2551 and TJA1050. The digital signals are on the left of the figure, while the analog signals are on the right. The three primarily blocks in this diagram are the 2.5V voltage source, the driver control and the receiver both interfacing between the digital and analog world.

The CANH and CANL voltages are determined by the driver control reacting to the digital transmit signal and by a voltage source of 2.5V. This voltage source is responsible of setting the CAN bus at 2.5V when a node is powered on. This value corresponds to the recessive state voltage. The 2.5V source is connected to each line through a high impedance resistor, 25 kΩ in the case of the TJA1050 [22]. The value of these resistors contributes to the busload and therefore the maximum number of nodes a CAN environment can support. For example, a high-speed CAN bus on which each node uses a MCP2551 CAN transceiver can support up to 112 nodes [20]. This source can be turned off by the power-on reset block to disconnect the CAN transceiver from the CAN bus.

To obtain an analog dominant bit level, the driver control achieves the bus voltage by driving two transistors. One is connected between the voltage supply (V_{DD} = 5V) and the CANH wire and acts as a pull-up**,** while the other one connects the CANL line to the ground (V_{SS}) and acts as a pull-down. Even though the block diagram shows two outputs on the driver control, both transistors are conducting simultaneously and are off at the same time.

The voltage drop through such a transistor and a diode is about 1V to 1.5V depending on their impedance and the current going through them. Assuming a supply voltage of 5V, when the transistor connected to CANH is conducting, CANH is pulled-up from 2.5V to only 3.5V due to a 1.5V drop between the 5V source and CANH caused by the transistor and the diode. When CANH is pulled-up, CANL that was idling at 2.5V is pulled-down and reaches 1.5V due to a voltage drop of 1V across its transistor and diode. The diodes between the lines and the transistors are also protection against high-voltage transients.

The receiver is basically a discriminator circuit having as inputs the CANH and CANL wires and for output the digital receive signal. A discriminator circuit compares the voltage between its two inputs and sets its output to 1 when both signal are the same, and to 0 when they are different. In other words, the digital output is based on differential voltage levels between the lines. Since interference is generally induced almost equally in both twisted wires, CANH and CANL, and knowing that the receiver reacts to a difference in voltage level between the two lines, this circuit provides high noise immunity.

Several other features are implemented in basic CAN transceivers [20]. The TXD dominant detect block is used for ground fault protection (equivalent to transmitting only dominant bits) on TXD input. The thermal shutdown block disables the driver control outputs controlling the transistors when they overheat. Transistor overheating can be caused when conducting an excessive current due to a short circuit on the bus. An external signal, Rs for Figure 32, can control the rise and fall times of CANH and CANL in order to reduce electromagnetic interference (EMI), also called radio frequency

interference (RFI). This signal also controls the CAN transceiver sleep mode. In sleep mode, the discriminator, or receiver, operates at a lower current and the driver control is turned off.

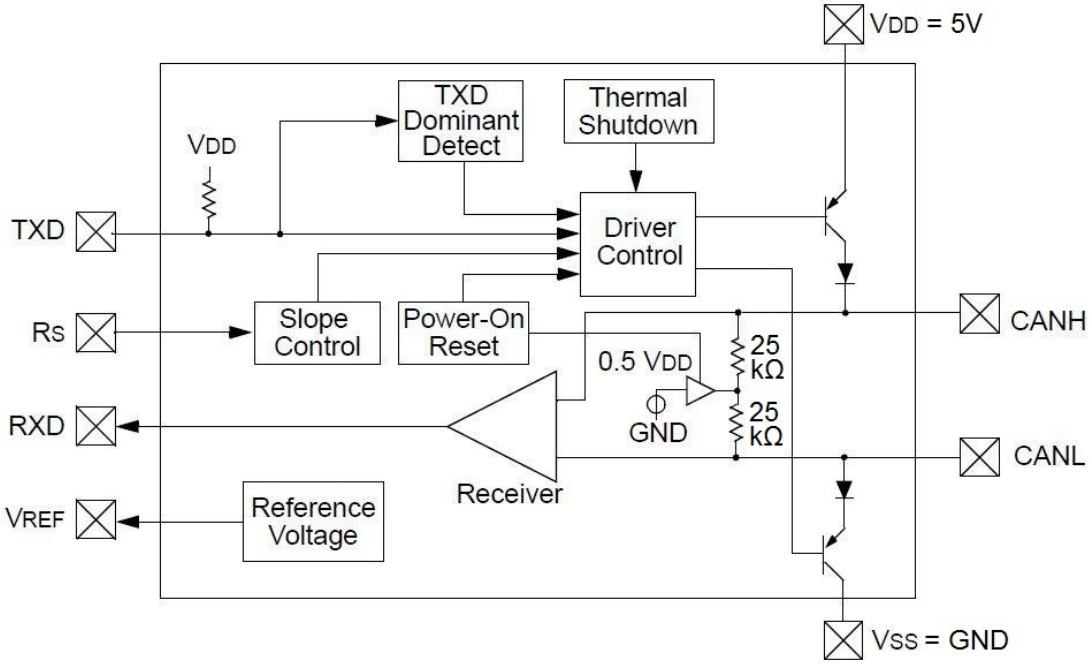


Figure 32. CAN transceiver MCP2551 block diagram [20].

Filters can be placed between the CAN transeiver and the CAN bus to help reduce noise. In the case of a single-wire CAN bus, EMI can be reduced by adding next to the CAN transceiver an inductor on the CAN wire. For a dual-wire CAN bus, high-speed CAN bus, a common mode choke and capacitor can be added to improve radiated emissions. In both cases, electrostatic discharge transient suppressor techniques, also known as ESD protection techniques, can be used to prevent permanent damage to the CAN transceiver related to undesired voltage transcients on the bus. One of these techniques involve the connection of back-to-back zener diodes, such as mmbz27vclt1 [40] or pesd24vs2uat [41], as close to the CAN transceiver as possible between CANH and the ground, and CANL and the ground. Other techniques use ESD capacitors or metal oxide varistors (MOV) or other suppression devices, instead of or in addition to back-to-back zener diodes. Providing a good ground to CAN transceivers and connecting it only at only one end of the cable to avoid ground loops, using a twisted pair wires to reduce high frequency noise, and using shielded cables (twisted pair and ground) in high RF environments are efficient noise reduction solutions easy to implement.

8. Use of Identifiers

In a CAN network, a technique sometimes used to easily identify a device is the division of the identifier into two fields: the device identifier and the message identifier. This can be useful when several similar devices are connected on the same network. Table 8 shows an example of a Tritium WaveSculptor [37] configuration using a 6-bit device identifier of "0b100 000x xxxx" and 5-bit message identifiers.

Table 8. Device and message identifiers.

Motor Controller TX ID: 0x400 + Message Identifier															
Device ID	Msg ID	ID	Device ID		Msg ID		Msg Name	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
			Bit 10	Bit 5	Bit 4	Bit 0										
32	0	0x400	0b100 000		0 0000		Id. Info.	8	Tritium ID				Serial Number			
32	1	0x401	0b100 000		0 0001		Status Info.	8	Limit Flags		Error flags		Act. Motor		Reserved	
32	2	0x402	0b100 000		0 0010		Bus Measur.	8	Bus Voltage				Bus Current			
32	3	0x403	0b100 000		0 0011		Velocity	8	Motor Velocity				Vehicle Velocity			
32	4	0x404	0b100 000		0 0100		Phase Current	8	Phase A Current				Phase B Current			

Some devices use the first byte of the data field as a sub-identifier, often referred to as a mode. When the identifier is not already divided in two, the first byte might also be called message identifier. This method usually requires more coding in the controller to decode the information. Table 9 details a battery management system (BMS) message having a fixed identifier of 0x600 and using sub-IDs.

Table 9. Sub-identifiers.

BMS TX ID: 0x600 + Sub-Identifier											
Sub ID	ID	Msg Name	DLC	Byte 0 Sub-ID	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0	0x600	Problem Flags	7	0x00	Error Cause		Active Error		Warning		
1	0x600	Master Measur.	8	0x01	Current		Voltage		SOC	Temp. Master	
2	0x600	Temperature	7	0x02	Average		Min		Max		

These techniques are not only useful to rapidly structure custom CAN buses, but are also standardized and used as a basis for communication protocols based on CAN.

9. CAN Message Definition

This section defines some common terms used to discuss CAN messages.

9.1 Message and Signal

Each identifier is assigned a maximum of 8 bytes (data field) referred to as a message, regardless of the division and order of the data. A message is usually broken into signals, which are subdivisions of the data field, and therefore of the message [5][25][26]. The only rule in creating a signal is: it has to be made of consecutive bits. Its length varies from one bit to a few bytes, and can overlap 2 bytes of the data field even if it is a 2 bit signal. There are no rules concerning the length of signals; however, most controllers do not allow signals longer than 32 bits. If a custom microcontroller is used, for instance a 16 bit microcontroller, signals should not exceed 16 bits to avoid extra coding for storage and arithmetic operations. Table 10 shows examples of a few messages and their signals.

Table 10. Message and signal examples.

Data Field (Message)							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Mode = 0	Info	Errors	Passives	Actives	Warnings	Limits	
Mode = 1	Batt. Status			Batt. Current		Batt. Voltage	
Mode = 2	State of Charge						
Mode = 3	Avg. Cell Temp.		Max. Cell Temp.		Avg. Cell Temp.		

9.2 Mapping and Positioning of Signals

The bit order, or bit order of significance, within a byte is always increasing from right to left as specified by the CAN communication protocol [1]. In other words, the most significant bit (msb) is always at the leftmost position of a byte and the least significant bit (lsb) at the rightmost position. The byte numbering, defined as the number assigned to each byte of the data field, is also fixed, but is increasing from left to right [24][25][26] [27]. The first byte sent is defined as byte 0 and the last byte is defined as byte 7. These concepts are illustrated in Table 11.

Table 11. Byte numbering and bit order.

Data Field (Message)							
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
msb ... lsb	msb ... lsb	msb ... lsb	msb ... lsb	msb ... lsb	msb ... lsb	msb ... lsb	msb ... lsb

Knowing that the bit order of significance and byte numbering are fixed, to map a CAN message and to position its signals in the data field the following, 4 pieces of information are required:

- Byte order of significance (endianness)
- Bit numbering
- Message progression (and overall bit numbering)
- Start bit of signals

It should be noted that the length of a signal is needed for its definition, but not to position it within the message.

9.2.1 Byte Order (Endianness)

When discussing byte order, 2 methods of sending the bytes of a message exist, little-endian and big-endian [26]:

- **Little-endian** means sending the least significant byte (LSB) first and the following bytes in increasing order of significance.
- **Big-endian** means sending the most significant byte first (MSB) first and the following bytes in decreasing order of significance.

These expressions are often referred to as Intel and Motorola respectively, since Intel processors use the little-endian method and Motorola processors use the big-endian one. Table 12 explains this nomenclature.

Table 12. Byte ordering.

Endianness	Referred to as	Byte order							
		Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Little-endian	Intel	LSB	...	...	...	...	...	...	MSB
Big-endian	Motorola	MSB	...	...	...	...	...	...	LSB

In summary, the expression little-endian and big-endian define the byte ordering of data, LSB to MSB and MSB to LSB respectively. In other words, they inform on the ordering of significance in the data field of a CAN message, but give no information on the numbering associated to each bit stuffed within a byte of the data field.

9.2.2 Bit Numbering

The bit numbering, indexing of bits within a byte, can be defined by the following 2 methods [26][28]:

- Decreasing from left to right (Sawtooth)
- Decreasing from right to left (Sequential or Monotone)

Table 13. Bit numbering.

Bit Numbering	Referred to as	Byte							
		msb	...	...	...	...	...	...	lsb
		Bit	Bit	Bit	Bit	Bit	Bit	Bit	Bit
Decreasing from left to right	Sawtooth	7	6	5	4	3	2	1	0
Decreasing from right to left	Sequential (Monotone)	0	1	2	3	4	5	6	7

The bit numbering, also called bit counting, is independent of the bit order, which is referring to the msb to lsb bit progression within a byte and is fixed in CAN communication as explained previously.

9.2.3 Message Progression

Now that the byte ordering, bit ordering, byte numbering and bit numbering are defined, the bit numbering progression from one byte to another, called message progression, must be defined. The message progression assigns a bit number from 0 to 63 to each bit in the message. The message progression can be [26][25]:

- Forward (numbering from the first byte)
- Backward (numbering from the last byte)

In other words, regardless of the bit numbering technique used, forward means the bit numbering begins from the first byte sent (bit 0 is in byte 0), and backward means the bit numbering begins from the last byte sent (bit 0 is in byte 7). In order to send and receive CAN data in a coherent manner, it is important for CAN users to know the bit number of each bit stuffed in a message, i.e. the overall bit numbering. This results in 4 possibilities, expanded from references [26][27][28]:

Table 14. Message progression.

Msg Progression	Bit Numbering	Data Field (Message)							
		Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Forward	Sawtooth	7...0	15...8	23...16	31...24	39...32	47...40	55...48	63...56
	Sequential	0...7	8...15	16...23	24...31	32...39	40...47	48...55	56...63
Backward	Sawtooth	63...56	55...48	47...40	39...32	31...24	23...16	15...8	7...0
	Sequential	56...63	48...55	40...47	32...39	24...31	16...23	8...15	0...7

Terminology used to define the bit numbering is discussed next. When representing 64 bits over the 8 bytes of the data field, the expression sawtooth seems to not only define the numbering within a byte, but also the message progression for the forward message progression. In a forward sawtooth message, the bit numbering increases from right to left while the message progression is from left to right creating discontinuities in the overall bit numbering. This expression kind of loses its sense for the big-endian byte order, since the bit numbering increases from right to left and the message progression is also from right to left creating no discontinuities as implied by the word “sawtooth”. The overall bit numbering of a backward sawtooth message has no discontinuities and looks like a forward “sequential” message starting the counting from the end of the message.

Being inconsistent with the continuous and discontinuous overall bit numbering, the bit numbering expressions are inadequate to visualize the overall bit numbering. This is one of the reasons why it is essential to specify the message progression in a message to avoid confusion from the user when defining the overall bit numbering.

9.2.4 Start Bit of Signals

As shown in Table 10, signals are variables, or pieces of data, contained by a message. They are the actual information, i.e. they can represent a voltage, a current, a state of charge, a temperature, a speed, a status, errors, warnings, etc. The start bit of a signal is usually its lsb when discussing CAN bus, however this is not always the case. Therefore, when positioning a signal in a message, the start bit used must be specified as the:

- lsb of the signal
- msb of the signal

If a signal is defined using the lsb as its start bit, the bit-wise progression is towards the left, but the byte-wise progression depends on the byte order.

9.2.5 Display Formats

This section details 6 display formats for messages using the terminology explained in the above sections. In an attempt to further clarify these display formats a column describing visually the bit-wise and byte-wise progression is added to Table 15 which summarizes the 6 common display formats in CAN communication, expanded from [26][24][28].

Table 15. Display formats.

Display Format	Byte Order	Bit Numbering	Msg Progression	Start Bit of signals	Bit progression (from the start bit to full length)
Intel Standard	Little-endian	Sawtooth	Forward	lsb	bit-wise: to the left byte-wise: to the right
Intel Sequential	Little-endian	Sequential	Forward	lsb	bit-wise: to the left byte-wise: to the right
Motorola Forward lsb	Big-endian	Sawtooth	Forward	lsb	bit-wise: to the left byte-wise: to the left
Motorola Forward msb	Big-endian	Sawtooth	Forward	msb	bit-wise: to the right byte-wise: to the right
Motorola Backward	Big-endian	Sawtooth	Backward	lsb	bit-wise: to the left byte-wise: to the left
Motorola Sequential	Big-endian	Sequential	Forward	msb	bit-wise: to the right byte-wise: to the right

Table 16 is a compact form of Table 15 using the 3 main CAN display formats.

Table 16. Compact form of the 3 main CAN display formats.

Display Format	Byte Order	Bit Numbering	Msg Progression	Start Bit of signals	Bit progression (from the start bit to full length)
Intel Standard	Little-endian	Decreasing from left to right	Forward	lsb	bit-wise: to the left byte-wise: to the right
Motorola Forward lsb	Big-endian				bit-wise: to the left byte-wise: to the left
Motorola Backward			Backward		

9.2.6 Examples

9.2.6.1 Representation in the 3 main display formats

In this example, adapted from reference [24], a signal is represented in the 3 main display formats and is defined by its start bit and length. The DLC of the message must be greater or equal to 7 and it is assumed that the start bit is the least significant bit of the signal in each of display formats.

Specifications: Start bit (lsb) = 11, Length = 9 bits.

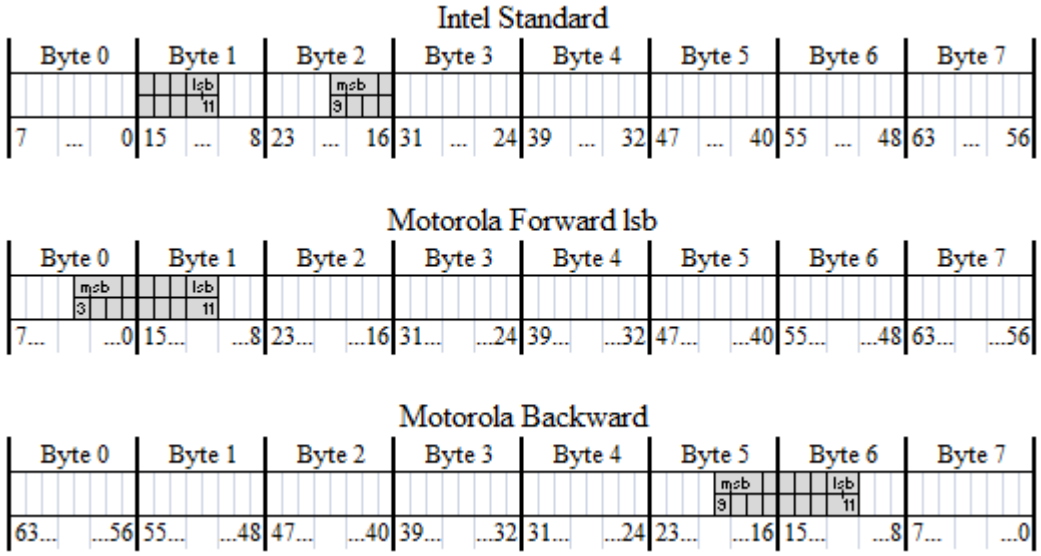


Figure 33. Representation of a signal in the 3 main CAN display formats.

This means that three controllers using the same signal specifications, but having three different display formats will send their signal in a completely different space of the data field.

9.2.6.2 Conversion from Intel Standard

As shown by the previous example, in order to decode a signal, it is essential to know its display format and apply an appropriate conversion when reading it with a controller using a different format. This example shows how a signal sent from controller using a Intel Standard display format is read by two controllers, one using a Motorola Forward lsb display format and the other Motorola Backward.

Specifications for Intel Standard: Start bit (lsb) = 11, Length = 9 bits.

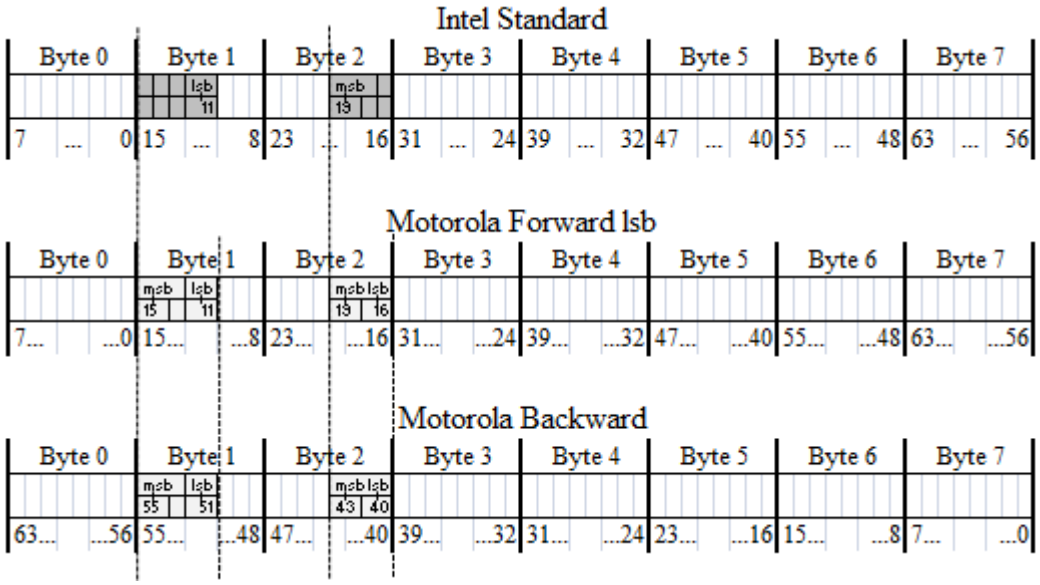


Figure 34. Conversion from Intel Standard.

Figure 34 shows how the signal in the Intel Standard display format is represented in the other two formats. In both cases, since the byte order is big-endian instead of little-endian, the signal must be rebuilt by concatenation of two signals. According to the little-endian byte ordering, the most significant part of the signal is in Byte 2 and the least significant part in Byte 1. Therefore, the concatenation is: signal in Byte 2 + signal in Byte 1. In the case of the Motorola Forward lsb, the start bit does not need to be converted, since it is using a forward message progression like the Intel Standard does. However, a conversion needs to be applied to the start bit for Motorola Backward, because the message progression used is backward. The graphical representation shown in Figure 34 is an easy method to find the new start bit in a different display format.

Specifications for Motorola Forward lsb: Start bit (lsb) = 11, Length = 9 bits
Specifications for Motorola Backward: Start bit (lsb) = 51, Length = 9 bits

For the most significant part of the signal, the start bit is always the first bit of the following byte and the length is the remainder of the total length not used by the least significant part of the signal. Thus, a detailed expression of the specifications for each display format is:

Specifications for Motorola Forward lsb:
[Start bit (lsb) = 16, Length = 4 bits] + [Start bit (lsb) = 11, Length = 5 bits]
Specifications for Motorola Backward:
[Start bit (lsb) = 40, Length = 4 bits] + [Start bit (lsb) = 51, Length = 5 bits]

The graphical representation of the conversion from Motorola Forward lsb and from Motorola Backward can be found in Appendix A.

9.3 Message and Signal Definitions

The difference between messages and signals was established in Section 9.1. This section describes in detail both messages and signals, and how to filter them in order to receive and send CAN messages.

9.3.1 Message Definition

A CAN message is defined by the following characteristics:

- Message name
- ID
- ID mask
- DLC (payload size)
- Payload filter (payload value)
- Payload mask
- Periodic interval (rate in ms)

The message name is the name of the message associated with a data frame or remote frame meeting the filtering characteristics: ID, ID mask, payload filter and payload mask. As described in Section 5.4 covering the control field of a message, the DLC can be between 0 and 8, and describes how many bytes of the data field are used for a given message. Messages can be sent when triggered by an event, such as a remote frame, or can be transmitted periodically. The periodic interval value is commonly expressed in milliseconds.

9.3.1.1 Filtering

CAN communication follows a producer-consumer model where all the nodes are masters. Messages give no information on the destination (consumer), only on the source (producer) [9]. Therefore, each node must read all the messages on the bus. When a message is sent (produced) by a node, it is up to the other nodes on the network to decide whether to receive it (consume) or to ignore it. In other words, nodes need to sort, or filter, the messages they are interested in receiving.

When using microcontrollers having CAN integrated features, also called CAN chips, filtering is done by hardware using CAN buffers and by software using a custom dispatcher. In microcontrollers without specific CAN buffers, sorting is accomplished by software only and is less efficient.

The sorting process has the two following filtering stages:

- ID filtering (using the ID and the ID mask)
- Payload filtering (using the payload filter and the payload mask)

In the first case, messages are filtered according to their identifier. The filter is the identifier mask defining which bits of the identifier to care about. The second type of filtering is used when a portion of the data field is used as a sub-identifier. Messages are sorted using a mask on the payload, named payload mask, indicating which bits of the data field to care about. Usually, the payload mask defines which bits of the data field to use as sub-ID allowing the demultiplexing of signals in messages using modes, as first

explained in Section 8. The payload filter is more likely the sub-ID of a message, but could be a specific payload value. It should be noted that the ID is to the ID mask what the payload filter is to the payload mask. Since the payload filtering is used to filter the sub-IDs of a specific ID, the ID filtering is processed first. When sub-IDs are not used, the payload mask is an all-pass filter. Table 17 presents an example of an ID filtering process while Table 18 details a payload filtering example using sub-IDs, based on reference [5].

Table 17. ID filtering example.

Filtering Configuration	Binary Value	Results
ID = 0x400	0b100 0000 0000	
ID mask = 0x7E0	0b111 1110 0000	1 = care, 0 = don't care
ID acceptance filter	0b100 000x xxxx	x = any value allowed
Incoming Message		
ID = 0x403	0b100 0000 0011	
ID acceptance filter	0b100 000x xxxx	Message accepted
ID = 0x404	0b100 0000 0100	
ID acceptance filter	0b100 000x xxxx	Message accepted
ID = 0x600	0b110 0000 0000	
ID acceptance filter	0b100 000x xxxx	Message ignored

Table 18. Payload filtering example.

Filtering Configuration	Binary Value	Results
ID = 0x600	0b110 0000 0000	
ID mask = 0x7FF	0b111 1111 1111	1 = care, 0 = don't care
ID acceptance filter	0b110 0000 0000	Only ID = 0x600 accepted
Payload filter = 0x01 00...00	0b00000001 00000000 ... 00000000	
Payload mask = 0xFF 00...00	0b11111111 00000000 ... 00000000	1 = care, 0 = don't care
Payload acceptance filter	0b00000001 xxxxxxxx ... xxxxxxxx	x = any value allowed
		Sub-ID = 0x01 in byte 0
Incoming Message		
ID = 400	0b100 0000 0000	
ID acceptance filter	0b110 0000 0000	Message ignored
Payload = 0x01 00 ... 00	-	
Payload acceptance filter	-	-
ID = 600	0b110 0000 0000	
ID acceptance filter	0b110 0000 0000	ID accepted
Payload = 0x03 00 ... 00	0b00000011 00000000 ... 00000000	
Payload acceptance filter	0b00000001 xxxxxxxx ... xxxxxxxx	Message ignored
ID = 600	0b110 0000 0000	
ID acceptance filter	0b110 0000 0000	ID accepted
Payload = 0x01 00 ... 00	0b00000001 00000000 ... 00000000	
Payload acceptance filter	0b00000001 xxxxxxxx ... xxxxxxxx	Message accepted

9.3.1.2 Post Office Analogy

As mentioned before, CAN messages are not destination oriented, but source based. However, most nodes need only a small subset of CAN messages travelling on the bus. Modules are required to filter the information transmitted on the bus to select the messages of interest. A relevant and simple analogy to understand the sorting of CAN messages is the post office analogy, adapted from reference [5].

Filtering, also called sorting, can be done by hardware and/or software. Microcontrollers featuring integreted CAN controllers typically have a hardware layer involving programmable buffers acting as a first filter. A custom dispatcher can be programmed in the target module's CAN chip software to refine the first sorting. The post office analogy terminology is defined in Table 19.

Table 19. Post Office Analogy

<table>
<tr><th colspan="2">Post Office</th><th colspan="2">CAN</th></tr>
<tr><td colspan="2">Letters</td><td colspan="2">CAN messages</td></tr>
<tr><td colspan="2">Postman</td><td colspan="2">Hardware/Software dispatcher</td></tr>
<tr><td rowspan="2">Mail box
(Desired letter)</td><td>Address
(of the sender)</td><td>ID Filtering
(ID + mask)</td><td rowspan="2">Slot
(Desired CAN msg)</td></tr>
<tr><td>Name
(of the sender)</td><td>Payload filtering
(Payload + mask)</td></tr>
<tr><td colspan="2">Doorbell</td><td colspan="2">Interrupt (or a function trigger)</td></tr>
</table>

Imagine a world where letters are CAN messages and the postman is the hardware and software dispatcher. In this world, the mail system works according to the sender's address and the name, instead of the destination information. Also, a house owns a mail box for each type of senders, in other words for each type of desired letters. Several people can send letters from the same address and a mail box is only limited to one sender's address, but can contain letters from numerous people from that address. A letter is desired when its sender's address and name match the ones on the mail box. In this analogy, a desired letter represents a filtered CAN message, i.e. a CAN message having an identifer and a data field fulfilling the criteria of the identifier filter and the

payload one. For priority mail, the postman notices the mail box, but delivers the urgent letter directly to the front door and rings the doorbell. This is the equivalent to setting a desired CAN message to an interrupt or a trigger function.

9.3.2 Signal Definition

A CAN signal is described by the following characteristics [7][26][27]:

- Signal name
- Start bit
- Length (in bits)
- Byte order (little-endian or big-endian)
- Data type (signed, unsigned, float, double, boolean, etc.)
- Units (mV, A, ºC, km/h, etc.)
- Scale (also called factor or gain)
- Offset
- Range (minimum and maximum)

9.3.2.1 Scaling

Signals are often converted before being transmitted over the CAN network. The purpose of the scale and offset of a signal is to convert the raw value of a signal to its engineering value. The physical value, or unit value, is the signal value having a scientific or physical meaning by being associated to a unit. The raw value, or bus value, is the signal value transmitted over the CAN bus. The scale and offset are adjusting the raw value of the signal to obtain its physical value expressed in the desired unit. The following basic linear equation is used to express the relation between the physical value and raw value, based on [25][27]:

$$physical\ value = (scale)(raw\ value) + offset$$
$$y = mx + b$$

10. Bus Configurations

The basic configuration of a CAN bus is the bus topology. Figure 35 illustrates this concept using devices found on an electric vehicle CAN network.

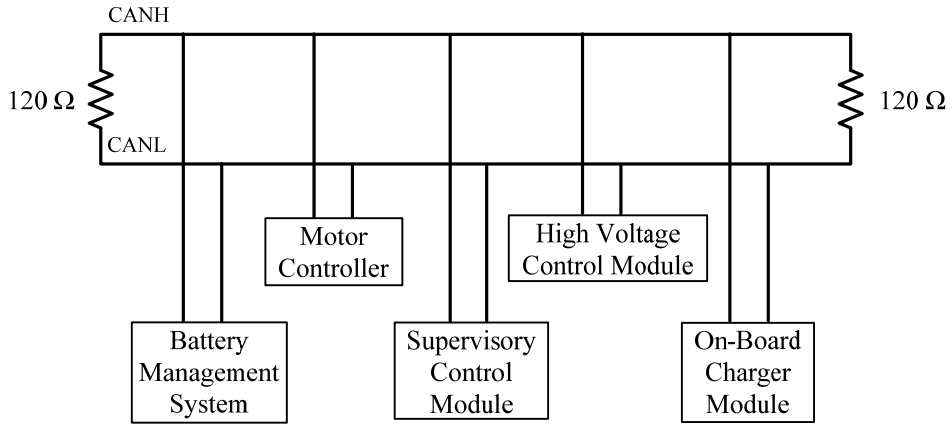


Figure 35. Bus topology.

In this topology, a main bus of 2 wires exists and devices are connected to it in parallel. The length of the wires connecting each device to the bus can vary. While this technique works well for small networks, electrical noise can be experienced on large networks, since every single branch connected to the main bus acts as an antenna.

In order to reduce the electrical noise, the two 120Ω termination resistors should be placed next to the 2 farthest controllers leaving only short leads between the resistors and the controllers, thus eliminating the two branches acting as antennas. This method is still considered a star topology. It retains simplicity and flexibility for the addition or removal of devices on the bus. Figure 36 gives an example of such a topology.

A bus topology can be defined as nodes connected by passive links through a single cable allowing transmission in both directions [19]. Hubs are generally used on the main bus to easily add and remove devices. They can be passive or active. Since the length of a CAN bus is short in automotive applications, hubs do not need to regenerate or amplify the bus signal, therefore only passive hubs will be considered. Active hubs require power and act as repeaters.

A star topology is a network where all the nodes are connected to a central one [19]. The addition of a hub on a bus configuration creates a node to which several nodes are connected. This node is therefore a centralized point for these nodes modifying the architecture of the network. When a hub is inserted to a bus structure, the network architecture becomes a combination of the bus and star topologies. Networks using more than one topology are called hybrid networks. A CAN network using one or more hubs, as shown on Figure 37, is considered to have a hybrid star-bus topology with a bus as the network backbone. This topology is sometime referred to as a cascaded star topolgy. A disadvantage of this technique is the addition of more branches, and therefore more noise in the network.

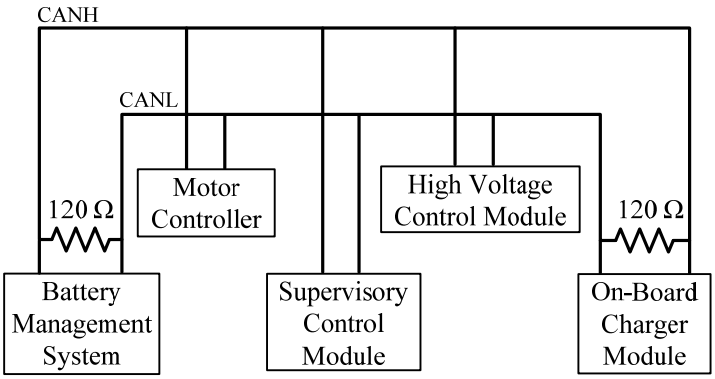


Figure 36. Bus topology minimizing noise.

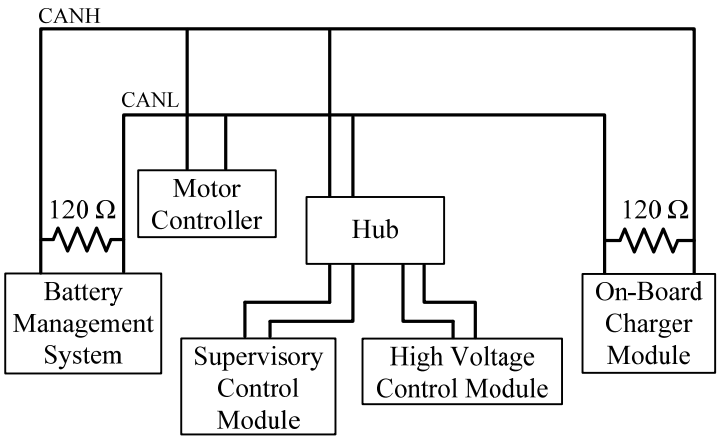
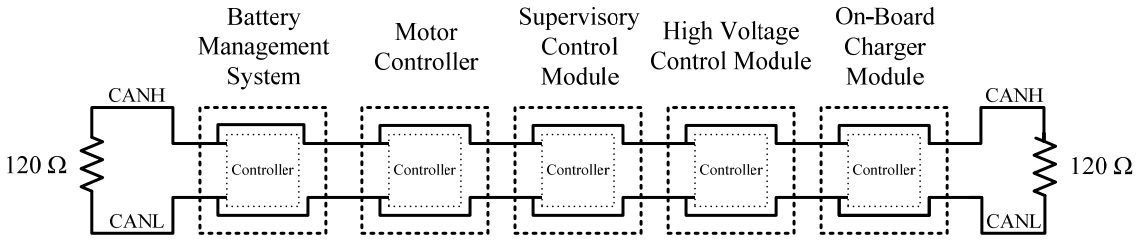


Figure 37. Hybrid star-bus topology.

To reduce the electrical noise to a minimum, another variation of the bus topology known as a daisy chain can be used. This method brings the main bus directly to every device on the CAN bus removing the branches, and therefore the antenna effect induced by them. In this configuration, the devices have an internal parallel connection design and each device acts as a hub for the following device in the chain. The expression daisy chain topology can be viewed as a bus topology with short studs. However, the expression is appropriate regarding the method used to create the wiring harness for the configuration and has an influence on the hardware of the devices connected to the bus. The absence of hubs in this configuration makes it more difficult to add and remove devices and makes the overall design more complex. Figure 38 illustrates this topology.



Note: The resistors can be internally integrated to the farthest devices' circuits.

Figure 38. Daisy chain with short studs.

Another method of creating a daisy chain is to simply twist two wires together and screw them in place into a single slot of a connector as shown on Figure 39. However, these types of connectors are not found in vehicles for reliability reasons.

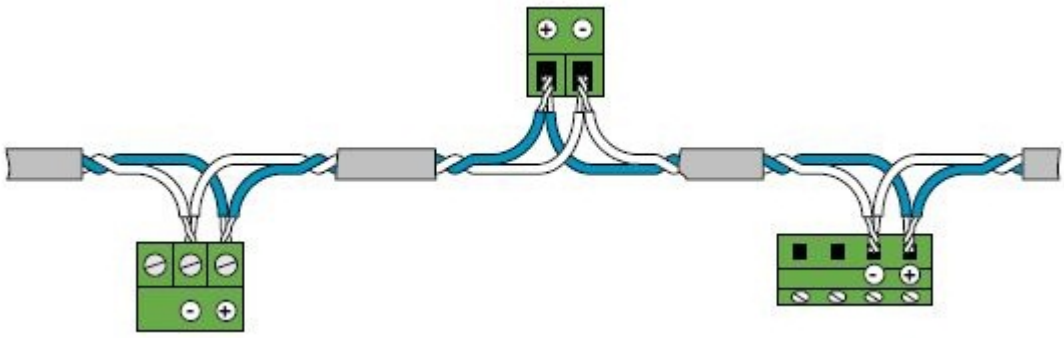


Figure 39. Daisy chain with twisted wires in the connector [31].

11. Commercial Devices

Some commercial devices are listed under “commercial controllers” in Table 7 where CAN transceivers, CAN protocol controllers, microcontroller with CAN and commercial controllers are enumerated. This section focuses not only on commercial controllers, but also on commercial devices. For the purpose of this document, commercial controllers are considered commercial devices, but the opposite is not true. A commercial device is the hardware portion of a CAN tool and requires a software application to interface and interact with the user. It is important to distinguish commercial devices from their software applications, since some sophisticated software can be used with different hardware while other less elaborate software is specific to a device.

Commercial devices for CAN can be categorized in 4 major groups:

- Scopes
- Loggers
- Controllers
- Hardware In the Loop (HIL)

Scopes are basically CAN to USB gateways allowing one to visualize the identifier and raw 8 bytes of data in a CAN message. A logger purpose is to log traffic on a CAN bus. Some can be programmed and operate with constant computer interaction, in other words they are stand alone devices with memory; whereas other devices have no memory but still allow logging through a software application by using the computer’s memory. As discussed in Section 7, controllers are programmable microcontrollers with circuitry allowing communication on one or several CAN buses. Hardware in the loop simulation devices are machines able to interact in real-time with an external system and update the measurements in a model running on a computer, or on the device itself, to test real-time embedded systems [38].

The list of commercial devices available on the market is extensive. Some of the most common ones are summarised in Table 20 with their features, software and vendor, info extracted from [43][44][45][46][47][48][49][50][51].

Table 20. Commercial devices.

Commercial Devices	Features					Software Applications (not exclusively)	Vendor
	Scope	Stand Alone Logger	Not Stand Alone Logger	Controller	HIL		
CANView USB	x		x			RM CAN-Device Monitor	RM
Leaf Light HS	x		x			Kvaser CanKing	Kvaser
CANcaseXL	x		x		x	Vector tools: CANoe	Vector
CANcardXL	x		x			Vector tools: CANalyser	Vector
neoVI	x		x		x	Vehicle Spy	Intrepid
CANLog 4		x				G.I.N. Configuration Program	Vector
CANcaseXL Log		x				Vector tools: CANoe	Vector
MotoTron	x			x		MotoHawk: Simulink, MotoTune	Woodward
Mico-Autobox	x		x	x		Control Desk, Simulink	dSPACE
PCI/PXI/PCMCIA	x		x		x	Labview: Simulink, NI VeriStand	NI
CAN-AC2-PCI	x		x		x	Matlab Simulink: xPC Target	Softing
dSPACE	x		x		x	Control Desk	dSPACE

12. Applications for Vehicles

So far, this document has covered the theory involved in the CAN bus protocol, the hardware required to support it, its bus configurations, how to define CAN messages in a non-ambiguous manner and introduced a few devices supporting CAN. Representing a rugged serial bus, CAN has a wide range of applications from providing a network in building and factory automation, to connecting controllers in ships, to its use in aircrafts for navigation systems and sensors, to elevators, forklifts and numerous other applications in industrial controls [3][8]. CAN bus is used as an in-vehicle communication network by most vehicle manufacturers in North America, Europe and Asia [3]. In this section, the focus will primarily be on CAN bus as an intra-vehicle communication environment. In addition, CAN will be compared to LIN, a cheaper communication system for vehicles.

12.1 Types of Applications

There are usually multiple CAN buses in vehicles. Up to 5 CAN buses have been seen, for example 4 high-speed buses and a single wire one. The types of applications differ whether it is a single-wire bus or a high-speed one.

12.1.1 Low Speed Applications

As defined in Table 1, 33.3 kb/s is the speed of a single-wire CAN communication system. Transmission rate is the primary limit of single-wire buses. Fault tolerant buses and high-speed buses configured with a low baud rate can also be used for non-time critical purposes; however, requiring only one wire and being slightly cheaper than high-speed and fault tolerant buses, single-wire buses are generally used for these applications. Table 21 provides a list of examples for low speed applications, expanded from reference [3].

Table 21. Low speed applications.

Dashboard	Instrument cluster panel
	Cabin temperature controls
	Light sensor
	Steering wheel electronics
	Entertainment controls
	Infotainment controls
	Air Conditioning controls
Door control	Mirror
	Window
	Door locks
Seat control	Seat position
	Seat heater
	Occupancy sensor
Roof control	Interior lights
	Visor lights
	Moon roof
Engine control	Fans
	Non-time critical sensors

12.1.2 High Speed Applications

For applications requiring near real-time communications, high-speed CAN buses are used. With the complexity of car controls constantly growing, it is not infrequent to find more than one high-speed bus in a car. Examples of applications using high-speed CAN buses are enumerated in Table 22, based on reference [3].

Table 22. High speed applications.

ECU programming
Diagnostic interface
Engine management
Electric motor controller
High voltage battery management
Adaptive Brake System (ABS)
Body controller
Accident avoidance system
Fuel system

12.1.3 Diagnostic Interface and ECU Programming

There are 2 other key applications of CAN in vehicles [3]:

- Diagnostic interface
- ECU programming

Most electronic control units (ECUs) save diagnostic information that can be sent on a CAN bus as diagnostic trouble codes (DTCs) to other ECUs and a diagnostic tester. The diagnostic tester generally connects to the vehicle's On-Board Diagnostic II (OBD II) port to read DTCs present on the CAN network in order to make a diagnostic.

Calibration of variables in ECUs and software updates are inevitable when developing a vehicle. ECUs can usually be programmed through CAN bus to simplify these tasks, since not every ECU of a vehicle is easily accessible and/or removable. This programming method has the advantage of requiring no modifications to the vehicle's hardware and can generally be done using the OBD II port.

12.1.4 Gateways between CAN buses

As mentioned earlier in this section, it is not uncommon to find multiple CAN buses interconnecting electronic control units (ECUs) within a vehicle. The need for having multiple CAN environments comes from the constantly increasing number of ECUs requiring more information, thus using more bandwidth [3]. When the bandwidth of a CAN network is saturated, it is common practice to add a new CAN bus to allow the addition of new controllers and features. These new ECUs present on the new CAN environment might need information transmitted on another CAN bus. This is why, most ECUs have multiple CAN ports, i.e. 2 to 4, and some ECUs are used as gateways between different CAN buses. Gateways are also required to interconnect ECUs from CAN buses using different high speed frequencies or to interconnect ECUs from a single-wire bus to a high-speed one.

12.2 Comparing LIN and CAN

Table 23 compares major characteristics of LIN and CAN.

Table 23. Comparing LIN and CAN [3].

Feature	LIN	CAN
Network topology	Bus	Bus
Number of wires	2	1
Maximum data rate	20 kbps	1 Mbps
Communications method	UART based	Controller based
Network access	Master-initiated transmission	Nondestructive
Node support	1 master, up to 15 slaves	64-128; typically limited by physical layer or higher-layer protocol

The number of nodes a CAN environment can support shown in Table 23 is different from the details provided by Table 1. The literature does not agree on a specific limit of nodes allowed on a CAN bus. Some documents define the maximum number of nodes as 30 [2], others as 128 [3].The important fact to remember is that the quantity of nodes supported by a CAN bus is dependant and limited by its load and data rate.

Regardless of these 2 criteria, a CAN network can support more nodes than a LIN environment. Being controller based, CAN is a more expensive solution compared to a universal asynchronous receiver-transmitter based local interconnect network, i.e. UART based LIN. However, according to Table 23, CAN is 50 times faster, more efficient, more flexible and more robust than LIN [3]. Moreover, LIN is only designed for automotive applications, whereas CAN has numerous uses beyond vehicles.

The data rate of a LIN environment might be 50 times slower than a high-speed CAN bus, but is similar to a single-wire one. Having cheaper hardware, LIN is often used as an alternative to single-wire CAN, used to control functions in a localized area. However, single-wire CAN networks are not limited to localized areas. In such applications, the LIN master node acts as gateway between the LIN network and a CAN environment to extend communications to other ECUs spread all over the vehicle. In other words, LIN is

often used in vehicles as a sub-bus via a LIN master node to manage devices in a localized area, such as the roof or a seat, providing several complementary networks to the main CAN buses. Figure 40 illustrates a typical application of a LIN sub-bus in a vehicle, while Table 24 lists common LIN localized support areas, based on reference [3].

Table 24. Typical LIN localized support areas.

Doors
Mirror control
Mirror switch
Window lift
Door locks
Engine
Sensors
Small motors
Roof
Moon-roof controls
Light Sensor
Interior light
Visor lighting
Seat
Occupancy sensor
Seat position motor
Seat heater
Steering column
Wheel tilt/position control motor
Cruise control switches
Wiper control
Turn control

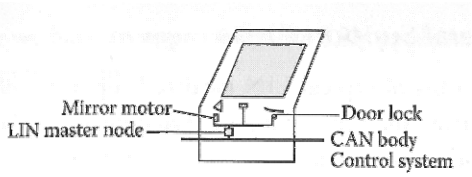


Figure 40. LIN sub-bus [3].

13. Conclusion

13.1 Summary

This work provides an overview on CAN protocol fundamental theory; a description of the hardware required to create a CAN environment; lists of devices supporting CAN available on the market; a rigorous definition of the nomenclature used to accurately define CAN messages and CAN signals. Different CAN bus configurations; as well as different techniques for using identifiers; and numerous automotive applications are covered.

13.2 Future Development

CAN Safety is a CAN-based technique providing safety in field bus systems. It is not related to secure communication where data encryption and decryption is used to protect systems from unauthorized access. It ensures the validity of CAN messages or the safety of the hardware in relation to explosions. This technology already exists, but is not commonly used in automotive applications. There are three types of CAN Safety technologies: safety-related communication, safety-critical communication and intrinsically safe communication [39]. Here there is a safe state the controller is forced into, given in any failure for a safety-related communication. This type of safety is found in CANopen Safety protocol and DeviceNet CIP Safety protocol; however a custom safety-related communication can be design as well. A safety-critical communication does not use a safe state, but redundancy instead. It can use redundant networks and/or redundant communication. Figure 41 shows an example of a safety-critical communication using a redundant communication [15]. The intrinsically safe communication is simply a CAN physical layer rated for certain conditions to ensure the CAN hardware will not cause any explosions. It finds applications in the petrochemical and chemical industries.

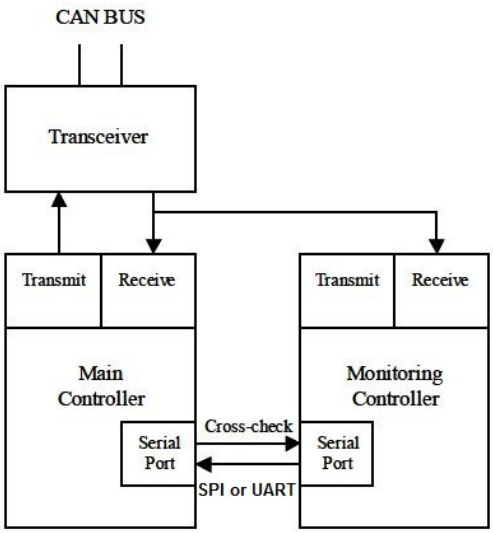


Figure 41. Safety-critical communication [15].

Even though this paper has explained the superiority of CAN over LIN, as technology evolves, a newer, faster and even more robust protocol is going to replace CAN in the near future. In the 21st century's first decade, a communication system called FlexRay was developed by the FlexRay Consortium [17]. In 2010, this group released the latest version of the protocol, the FlexRay™ Communications System specifications Version 3.0.1 [17]. This new robust serial networking technology designed for advanced control applications in the automotive industry is a time-deterministic, scalable and fault-tolerant protocol having a data rate up to 10 Mb/s [16][18]. Like most new technologies reaching the market, FlexRay is more expensive than older similar technologies such as LIN and CAN.

References

[1] Bosch. *"CAN Specification"*, Version 2.0, Robert Bosch GmbH, 1991.

[2] Kadionik, Patrice. *"Le bus CAN"*, École Nationale Supérieure Électronique Informatique & Radiocommunications Bordeaux, 2001.

[3] Held, Gilbert. *"Inter- and intra-vehicle communications"*, Auerbach Publications, 2008.

[4] Provencher, Hugo. "*Introduction au protocole de communication CAN"* [Presentation], ELE4202 Commande des processus industriels, Département de génie électrique, École Polytechnique de Mtl, Automne 2009.

[5] MotoHawk Training. *"CAN (Controller Area Network)"* [Presentation], Woodward MotoTron Control Solutions, 28 October 2008.

[6] Schultz, Katrina. *"GMLAN In-Vehicle Communication Networks"* [Presentation], General Motors, 1 October 2010.

[7] Linton, Becky. *"CAN Communication for NI Dynamic Testing Software (Hardware in the Loop)"* [Presentation], National Instruments.

[8] http://www.can-cia.org/, visited on 13th December, 2011.

[9] CAN in Automation. "*Controller Area Network (CAN)*" [Website], http://www.can-cia.org/index.php?id=systemdesign-can

[10] CAN in Automation. "*CAN physical layer*" [Website], http://www.can-cia.org/index.php?id=systemdesign-can-physicallayer

[11] CAN in Automation. "*CAN history*" [Website], http://www.can-cia.org/index.php?id=systemdesign-can-history

[12] CAN in Automation. "*CANopen*" [Website], http://www.can-cia.org/index.php?id=systemdesign-canopen

[13] CAN in Automation. "*DeviceNet*" [Website], http://www.can-cia.org/index.php?id=48

[14] CAN in Automation. *"CAN made easy Basic information on the CAN physical and data link layer"*, CiA, retrieved December 13th,2011, from www.can-cia.org/pg/can/additional/can_basics_print.pdf.

[15] Passemard, Michel. "*Atmel Microcontrollers for Controller area Network (CAN)*", Atmel Corporation, 4069A-CAN-02/04, http://www.atmel.com [Website].

[16] National Instruments. *"FlexRay Automotive Communication Bus Overview"* [Website], http://zone.ni.com/devzone/cda/tut/p/id/3352, visited on 28th March, 2012.

[17] FlexRay™. "About FlexRay™" [Website] http://www.flexray.com, visited on 28th March, 2012.

[18] www.freescale.com/files/microcontrollers/doc/fact_sheet/FLEXCOMMSYSTM4FS.pdf [Website].

[19] http://en.wikipedia.org/wiki/Network_topology, visited on 10th December, 2011.

[20] Microchip. "*MCP2551 High-Speed CAN Transceiver*" [datasheet], Microchip Technology Inc., 2007, DS21667E, [Website] http://www.microchip.com/.

[21] Microchip. *"dsPIC33FJXXXGPX06/X08/X10 Data Sheet High-Performance, 16-Bit Digital Signal Controllers"* [datasheet], Microchip Technology Inc., 2007, DS70286A, http://www.microchip.com/ [Website].

[22] Philips Semiconductors (NXP Semiconductors). *"TJA1050 High speed CAN transceiver"* [datasheet], Product specification, 22nd October 2003, www.nxp.com/documents/data_sheet/TJA1050.pdf

[23] Tech Note, *"CAN Data Field Format – Intel, Motorola Forward and Motorola Backward"* [Website], www.warwickcontrol.com, visited on 13th December, 2011.

[24] T. McLaughlin, Richard. *"CAN Data Formats and X-Analyser Functions"* [Presentation], 2011, Warwick Control Technologies, retrieved December 13th, from www.testing-expo.com/europe/05txeu_conf/pres/mclaughlin.pdf.

[25] Vector Academy, "*Introduction to CANoe / DENoe*" [video], v7.0, 2009-03-10, CANoe v7.0, Vector CANtech Inc.

[26] Vector CANdb++ Online Help, *"Mapping and Position Numbering of Signals in Frames"* [Software], Vector CANdb++ Editor, Version 3.0.62 (SP6).

[27] Matlab Product Help. *"CAN Unpack"* [Software], CAN Unpack :: Block Reference (Vehicle Network Toolbox™), Library: CAN Communication, Matlab 2009b.

[28] http://www.race-technology.com/wiki/index.php/CANInterface/ByteOrdering, visited on 13th December, 2011.

[29] SAE, "*Vehicle Architecture For Data Communication Standards"* [Website], http://www.sae.org/servlets/works/documentHome.do?comtID=TEVEES12, visited on 30th March 2012.

[30] National Instruments, (visited in February 2011), website: http://www.ni.com/pxi/ [Website].

[31] Wiring Controlsoft RS-485 networks, KeyMaster Systems, Hardware Installation Guidelines, Controlsoft, 2002, retrieved from www.powerrichsystem.com/Downloads/RS-485wiringNetworks.pdf

[32] ISO Committe Draft. "*Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*", ISO/CD 11898-1, ISO/TC 22SC 3 N, October 1999.

[33] ISO Committe Draft. "*Road vehicles – Controller area network (CAN) – Part 2: High-speed medium acces unit*", ISO/CD 11898-2, ISO/TC 22SC 3 N, September 1999.

[34] ISO Committe Draft. "*Road vehicles – Controller area network (CAN) – Part 3: Fault tolerant medium access unit*", ISO/CD 11898-3, ISO/TC 22SC 3 N, 2001-06-07.

[35] Wikipedia. "*On-board diagnostics*" [Website], http://www.en.wikipedia.org/wiki/On-board_diagnostics, visited on 28th March, 2012.

[36] Microchip. *"Section 21. Enhanced Controller Area Network (ECAN™)"* [datasheet], Microchip Technology Inc., 2007, DS70185A, http://www.microchip.com/ [Website].

[37] Tritium. "WaveSculptor CAN Bus Communications Protocol Specification" [Datasheet], 20 August 2007, http://www.tritium.com.au/index.html [Website].

[38] Provencher Hugo. "Embedded Real-Time System Development for Electric Vehicles", Faculty of Engineering and Applied Sciences, UOIT, ENGR 5910G Embedded Real-Time Control Systems, 2011-02-16.

[39] CAN in Automation. "*Safety in field bus systems*" [Website], http://www.can-cia.org/index.php?id=50

[40] ON Semiconductor. "***MMBZ15VDLT1G, MMBZ27VCLT1G, SZMMBZ15VDLT1G, SZMMBZ27VCLT1G** 40 Watt Peak Power Zener Transient Voltage Suppressors*" [Datasheet], Semiconductor Components Industries, LLC, 2012, http://www.onsemi.com/pub_link/Collateral/MMBZ15VDLT1-D.PDF

[41] NXP Semicondutors. "***PESDxS2UAT series** Double ESD protection diodes in SOT23 package*" [Datasheet], 2004 February 18, http://www.nxp.com/documents/data_sheet/PESDXS2UAT_SER.pdf

[42] Vector. "*Basic Vehicle Networks – An Introduction to CAN*" [Webinar], Vector CANtech Inc., 2012.

[43] RM Michaelides software & elektronik. "*Connecting PCs to mobile or stationary CAN networks*" [Website], http://www.rmcan.com/index.php?id=73&L=1

[44] Kvaser Advanced CAN Solutions. "*Home\Products\CAN\USB*" [Website], http://www.kvaser.com/en/products/can/usb.html

[45] Vector. "*VN1600*" [Website], http://www.vector.com/vi_vn1600_en.html

[46] Vector. "*CANlog 3 and CANlog 4*" [Website], http://www.vector.com/vi_canlog_en.html

[47] New Eagle. "*Welcome to New Eagle Learning Center*" [Website], http://www.neweagle.net/support/wiki/index.php?title=Main_Page

[48] dSPACE. "*MicroAutoBax II*" [Website], http://www.dspace.com/en/inc/home/products/hw/micautob.cfm

[49] National Instruments. "*Controller Area Network (CAN)*" [Website], http://www.ni.com/can/

[50] Softing. "*CAN bus Interface Card: CAN-AC2-PCI*" [Website], Softing your connection to excellence, http://www.softing.com/home/en/automotive-electronics/products/can-bus/interface-cards/can/pci-2.php

[51] Intrepid Control Systems, inc. "*neoVI FIRE : 6x CAN, 4x LIN - neoVI RED : 2x CAN, 2x LIN*" [Website], http://intrepidcs.com/neovifire/index.html

[52] Vector. "*Introduction to CAN*" [Website], http://www.vector-elearning.com/vl_einfuehrungcan_portal_en.html

[53] Vector. "*Vector Training Worldwide*" [Website], http://www.vector.com/vi_training_en.html

Appendix A

Conversion of Display Format

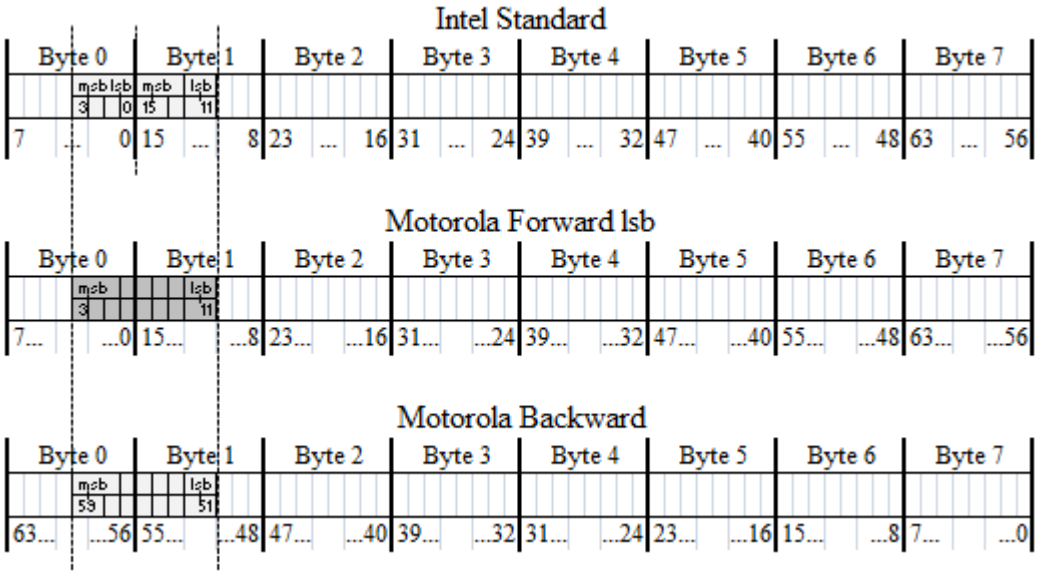


Figure 42. Conversion from Motorola Forward lsb.

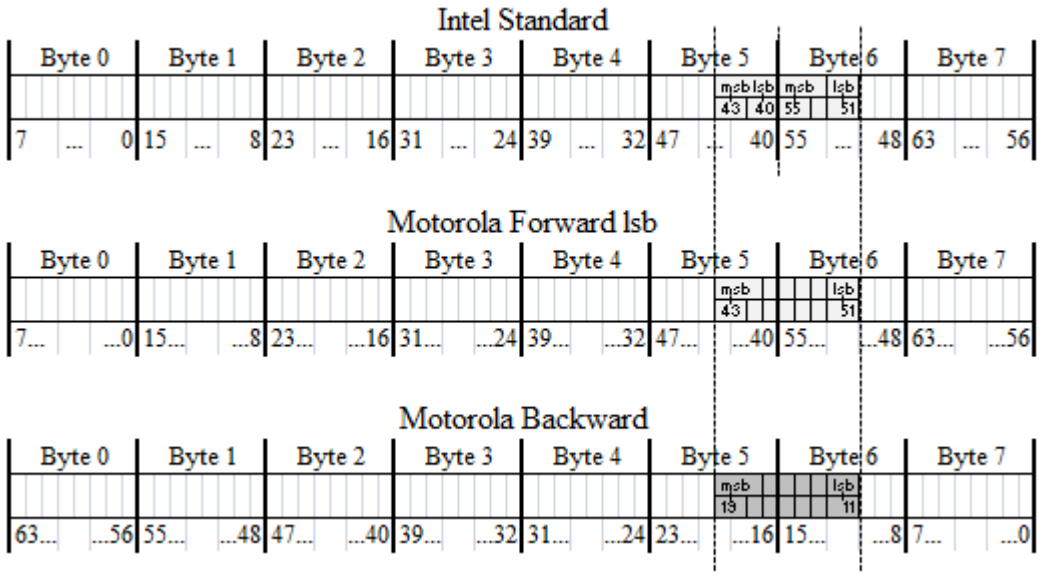


Figure 43. Conversion from Motorola Backward.