IMPLEMENTATION OF CONTROLLER AREA NETWORK FOR AUTOMOTIVE APPLICATIONS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

VLSI Design and Embedded System

By

LESLIN VARGHESE

Roll No: 20607008

Department of Electronics & Communication Engineering

National Institute of Technology

Rourkela

2008

IMPLEMENTATION OF CONTROLLER AREA NETWORK FOR AUTOMOTIVE APPLICATIONS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

VLSI Design and Embedded System

By

LESLIN VARGHESE

Roll No: 20607008

Under the Guidance of

Prof. K. K. MAHAPATRA

Department of Electronics & Communication Engineering

National Institute of Technology

Rourkela

2008

NATIONAL INSTITUTE OF TECHNOLOGY

ROURKELA

CERTIFICATE

This is to certify that the thesis report titled "**Implementation of Controller Area Network for Automotive Applications**" submitted by **Mr. Leslin Varghese** (Roll No: 20607008) in partial fulfillment of the requirements for the award of Master of Technology Degree in Electronics and Communication Engineering with specialization "**VLSI Design and Embedded System**" during session 2007-2008 at National Institute Of Technology, Rourkela (Deemed University) and is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any Degree or Diploma.

Prof. K. K. Mahapatra

Date: **Department of E.C.E**

National Institute of Technology

Rourkela-769008

ACKNOWLEDGEMENTS

It is a pleasure to thank many people who made this thesis possible.

I would like to take this opportunity to express my gratitude and sincere thanks to my supervisor **Prof. K. K. Mahapatra** for his guidance, insight, and support he has provided throughout the course of this work.

I am grateful to our teachers **Prof. G.S. Rath**, **Prof. G. Panda, Prof. S.K. Patra** and **Dr. S. Meher**. From these teachers I learned about the great role of self-learning and the constant drive for understanding emerging technologies, and a passion for knowledge.

I am thankful to Mr. Sudeendra Kumar for his encouragement and great support.

My special thanks go to PhD scholars, research scholars, friends, especially to Embedded and VLSI II lab friends, and juniors at NIT Rourkela for their encouragement and help throughout the course.

I would like to thank all faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela for their extreme help throughout course.

Finally, I am forever indebted to my parents, brother and sister for their prayer, love, understanding, endless patience and encouragement when it was most required. I truly believe that by giving thanks to all I mentioned above, I am expressing my gratitude to God who helped me in all my need.

Leslin Varghese

CONTENTS

ABSTRACT iv
LIST OF FIGURES v
LIST OF TABLES vii
INTRODUCTION 1
1.1 Can History 2
1.2 Can Overview 3
1.3 Anti-Lock Braking System Overview 4
1.4 Adaptive Cruise Control Overview 4
1.5 Fuzzy Logic For Embedded System 5
1.6 Problem Being Addressed 5

SERIAL COMMUNICATION USING CONTROLLER AREA NETWORK PROTOCOL 7
2.1 An Introduction To Communication Protocols 8
2.2 Can Protocol 9
2.3 Can Data Link Layer 11
2.3.1 Bus Arbitration 11
2.3.2 Can Message Transfer 12
2.3.3 Bit Stuffing 20
2.3.4 Error Process 21
2.3.5 Acceptance Filtering And Message Validation 25
2.4 Can Physical Layer 26
2.4.1 Bit Representation 27
2.4.2 Bit Encoding 27
2.4.3 Bit Timing And Sychronization 28
2.4.4 Bit Rate And Bit Length 30

MOTOROLA MC9S12DP256B 31
3.1 Introduction 32

3.2 Mc9s12dp256b 32
3.3 Features 32
3.4 Cpu (Star 12) 33
3.5 Operating Modes 35
3.6 Some On-Chip Resources 35
3.6.1 Enhanced Capture Timer 35
3.6.2 Analog To Digital Converter 38
3.6.3 Real Time Interrupt 40
3.7 Motorola Scalable Can: 41
3.7.1 Features 41
3.7.2 Transmit Structure 42
3.7.3 Receive Structure 42
3.7.4 Identifier Acceptance Filter 42
3.8 Hcs12 Fuzzy Engine 43

IMPLEMENTATION 46
4.1 Introduction 47
4.2 Vehicle Setup 47
4.3 Sensors 48
4.3.1 Wheel Speed Sensor 48
4.3.2 Distance Sensor 49
4.4 Speed Control Circuit For Dc Motor 49
4.5 Can Bus 50
4.5.1 Transceiver (Mc3388d) 51
4.5.2 Can Control And Status Registers 52
4.6 Anti-Lock Braking System 54
4.6.1 Fuzzy Block 54
4.6.2 Fuzzy Rules 55
4.7 Adaptive Cruise Control 56
4.7.1 Fuzzy Block 57
4.7.2 Fuzzy Rules 57
4.8 Seat Belt Module 58
4.9 Message Communicated 58
4.10 Results 59

CONCLUSION AND FUTURE WORK 64
5.1 Conclusion 65
5.2 Future Work 65

REFERENCES 66

ABSTRACT

The Controller Area Network (CAN) is a serial, asynchronous, multi-master communication protocol invented mainly for connecting electronic control modules in automotive applications needing high levels of data integrity and data rates of up to 1 Mbit/s. In reality the field of application of CAN is not only limited to automotive applications nowadays. It is now using in fields like industrial automation, medical equipments, and home automation. The objective of this work is to implement a CAN network for some real time control system. Here the system consisting of three application nodes; Anti-Lock braking system, Adaptive Cruise Control, and seat belt module, which are crucial parts in today's vehicles.

The hardware part of this work consists of a simple dc motor controlled electric vehicle, three MC9S12DP256B microcontroller modules as three nodes, and a stepper motor arrangement for controlling the speed. The CAN Network is implemented using the CAN modules available in the microcontroller as on chip peripheral. An external transceiver MC33388 is using for each node to drive the network. The ABS and ACC algorithms are implemented using the fuzzy logic support of the microcontroller. The operation of the system is as follows: If any sudden brake applied to the brake pedal the ABS module will be activated. By monitoring the slip ratio and brake pressure applied ABS controls the vehicle speed. In the meantime it will send a message to the seat belt module to activate the system. If the ACC module switched on, it will continuously monitors any forward obstacle is there in the front line of the vehicle, and if it detects any the module will inform the ABS module to take the necessary action.

In this thesis work main concern is to implement the CAN network to a more complex real time systems. The CAN network can apply to even more complex control systems by increasing its nodes. Depending on the application different CAN nodes have different CAN messaging needs. By using improved versions of CAN like, flexRay CAN, Time Triggered CAN (TTCAN), and by decentralizing the network again with sub network protocols like LIN we can achieve different application needs.

LIST OF FIGURES

Fig 1.1	Communication using CAN	3
Fig 2.1	Representation of a distributed network	9
Fig 2.2	CAN protocol ISO/OSI layered model	10
Fig 2.3	CAN bus arbitration	12
Fig 2.4	CAN data frame	13
Fig 2.5	CAN remote frame	16
Fig 2.6	Active error frame	17
Fig 2.7	Passive error frame	18
Fig 2.8	Overload frame	18
Fig 2.9	Interframe space for error active nodes	19
Fig 2.10	Interframe space for error passive nodes	19
Fig 2.11	Bit stuffing	20
Fig 2.12	CAN error states	23
Fig 2.13	CAN network	26
Fig 2.14	CAN dominant and recessive states	27
Fig 2.15	CAN bit time	29
Fig 3.1	Register set for CPU12	34
Fig 3.2	Timer module clock selection circuit and registers	36
Fig 3.3	Input capture functional block diagram	37
Fig 3.4	Basic output compare functional diagram	37
Fig 3.5	Clock chain for RTI	40
Fig 3.6	MSCAN block diagram	41
Fig 3.7	Block diagram of MSCAN fuzzy logic support	43
Fig 4.1	Vehicle setup	48
Fig 4.2	Code wheel sticker	48
Fig 4.3	Infrared sensor functioning	49
Fig 4.4	DC motor speed control circuit	50
Fig 4.5	CAN network	51
Fig 4.6	Transceiver pin connection	51
Fig 4.7	Jumper 15	52
Fig 4.8	Jumper 16	52
Fig 4.9	Input membership functions	55

Fig 4.10 Output membership function for ABS 56
Fig 4.11 Output membership function for ACC 57
Fig 4.12 Message $BB data frame 60
Fig 4.13 Message $DF data frame 61
Fig 4.14 Message $D2 data frame 62
Fig 4.15 Message $88 data frame 63

LIST OF TABLES

Table 3.1	Operating modes of STAR 12 CPU	35
Table 4.1	2-phase switching scheme of stepper motor	57
Table 4.2	Message communication chart	59

CHAPTER 1

INTRODUCTION

Electronics – we define broadly as being systems based on digital hardware and software – has gained central importance for many new automotive applications and services. On the production side we observe that the cost for electronics and IT is approaching the 50% threshold of all manufacturing costs. Perhaps more importantly, there are estimates that already today more than 90% of all vehicle innovations are centered around software and hardware (admittedly not only digital hardware, though). Electronic systems in cars can roughly be classified into three main areas:

1. Basic car functions, e.g., engine control, steering, and braking.
2. Secondary car functions, e.g., window control and immobilizers.
3. Infotainment applications, e.g., navigation systems, music and video entertainment, and location-based services.

For a good overview on the possible future of Vehicle electrical systems, is referred to almost all such applications are realized as embedded systems, that is, as devices which incorporate a microprocessor. The devices range from simple control units based on 8-bit micro controllers to infotainment systems equipped with high-end processors whose computing power approaches that of current PCs. The number of processors can be 80 or more in high-end cars. In a typical automobile the devices are connected by several separate buses.

Not surprisingly, many classical electronics and software technologies are already well established within the automotive industry, for instance hardware-software co-design, software engineering, software component re-use, and software safety.

1.1 CAN HISTORY

Controller Area Network (CAN) is a shared serial bus communication protocol, originally developed in 1986 by Robert Bosch GmbH. The increasing number of distributed control systems in cars and the increasing wiring costs of car body electronics led to the birth of the "Automotive Serial Controller Area Network" protocol. Although initially developed for use in the automotive industry, its use quickly spread to a wide variety of embedded systems applications like industrial control where high speed communication is required. With growing acceptance in various industries not necessarily related to the automotive industry, the protocol was renamed the Controller Area Network (CAN).

1.2 CAN OVERVIEW

A CAN system sends messages using a serial bus network. With every node connected to every other node in the network, the need for a central controller for the entire network is made redundant. A block diagram of a typical CAN network used for communication is shown in Fig 1.1.

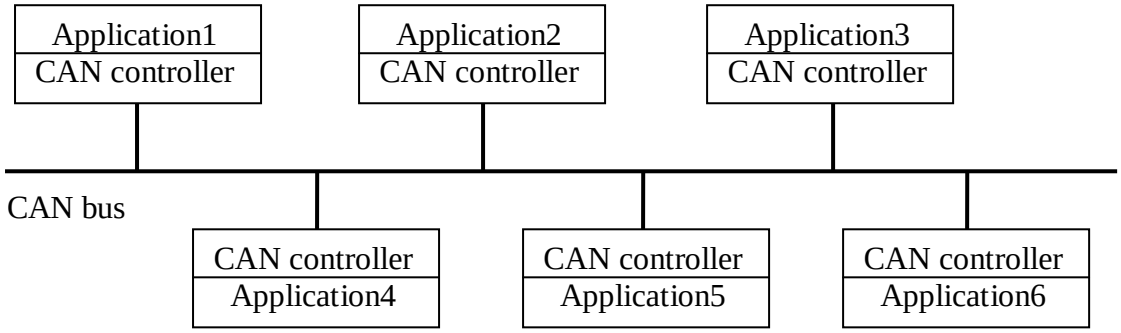


Fig 1.1 communication using CAN

CAN being a multimaster communication protocol; every node in the system is equal to every other node. Any processor can send a message to any other processor. Even if some processor fails, the other systems in the machine will continue to work properly and communicate with each other. Any node on the network that wants to transmit a message waits until the bus is free. Every message has an identifier, and every message is available to every other node in the network. The node selects those messages that are relevant and ignores the rest.

The CAN protocol was designed for short messages, no more than eight bytes long. The protocol never interrupts an ongoing transmission, but it assigns priorities to messages to prevent conflicts and to make sure that urgent messages are delivered first. The CAN protocol includes robust error detection and fault confinement mechanisms to make the traffic highly reliable. The robustness of the network is further augmented by the fact that any faulty node can be removed from the network without affecting the rest of the network. Unlike a traditional network such as USB or Ethernet, CAN does not send large blocks of data point to point from node A to node B under the supervision of a central bus master. In a CAN network, many short messages are broadcast to the entire network, which allows for data consistency in every node of the system.

The communication across a CAN bus starts with the application providing the CAN controller with the data to be transmitted. The CAN controller provides an interface between the application and the CAN bus. The function of the CAN controller is to convert the data provided by the application into a CAN message frame fit to be transmitted across the bus. A transceiver receives the serial input stream from the controller and converts it into a differential signal. These differential signals are transmitted physically across the CAN bus.

CAN transmits signals on the CAN bus which consists of a CAN High and CAN Low. These 2 buses carry signals of opposite polarity to overcome noise interruption. CAN use a bit arbitration technique in which the priority of accessing the bus is determined by the 11bit identifier. Due to the architecture a dominant bit will always override a recessive bit; the node with a lower identifier will have higher priority.

Every CAN Controller in a network will receive any message transmitted on the bus. Based on a filtering mechanism, the controller decides if the received information is relevant to the interfacing application and then proceeds to process the information.

1.3 ANTI-LOCK BRAKING SYSTEM OVERVIEW

The is to generate in real time the largest possible brake force while keeping the vehicle maneuverable and avoiding excessive wheel slippage. The absence of sensors capable of detecting on-line measurement variables, such as tire slip ratio and road condition, forces the ABS system designer to seek means to infer the road condition from the few available sensory data. The most important process parameters affecting the quality of control are the coefficient of friction between tire and road, the tire slip ratio, and the vertical force on the wheel. Generally, a high degree of uncertainty is associated with the measurement of the slip ratio between wheel and road surface. This problem is compounded by the sometimes rapid variation of road conditions with its attendant large variations of friction coefficients, slip ratio and vertical contact force between tire and road surface. The plant to be controlled (elastically suspended wheel, braking servo system, actuator) comprises significant transportation delays which limit the frequency response of the controller.

1.4 ADAPTIVE CRUISE CONTROL OVERVIEW

ACC is an automotive feature that allows a vehicle's cruise control system to adapt the vehicle's speed to the traffic environment. A radar system attached to the front of the

vehicle is used to detect whether slower moving vehicles are in the ACC vehicle's path. If a slower moving vehicle is detected, the ACC system will slow the vehicle down and control the clearance, or time gap, between the ACC vehicle and the forward vehicle. If the system detects that the forward vehicle is no longer in the ACC vehicle's path, the ACC system will accelerate the vehicle back to its set cruise control speed. This operation allows the ACC vehicle to autonomously slow down and speed up with traffic without intervention from the driver. The method by which the ACC vehicle's speed is controlled is via engine throttle control and limited brake operation.

1.5 FUZZY LOGIC FOR EMBEDDED SYSTEM

An objective of fuzzy logic has been to make computers think like people. Fuzzy logic can deal with the vagueness intrinsic to human thinking and natural language and recognizes that its nature is different from randomness. Using fuzzy logic algorithms could enable machines to understand and respond to vague human concepts such as hot, cold, large, small etc. It also could provide a relatively simple approach to reach definite Conclusions from imprecise information.

Almost every application, including embedded control applications, could reap some benefits from fuzzy logic. Its incorporation in embedded system could lead to enhanced performance, increased simplicity and productivity, reduced cost and time to market along with other benefits. Fuzzy logic has the advantage of modeling complex, non linear problems linguistically rather than mathematically and using natural language processing. The use of fuzzy logic requires, however the knowledge of human expert to create an algorithm that mimics his/her expertise and thinking. Also, studying the stability of fuzzy system is a demanding task.

1.6 PROBLEM BEING ADDRESSED

The ultimate problem being addressed is the design and development of a CAN network for crucial Automotive control systems like Anti-lock brake system (ABS) and Adaptive Cruise Control (ACC). This work demonstrated 3 node CAN network, one module is ABS, another ACC and third seat belt control. CAN is highly successful and reliable protocol in Vehicle networking and famous for its performance in all the aspects. The fuzzy logic support of microcontroller is utilized in this work by providing Fuzzy logic based

techniques to have proper control of ABS and ACC. Total setup of experiment executed using Freescale MC9S12DP256B microcontroller board as CAN nodes and it also includes a simple dc motor based electric vehicle.

CHAPTER 2

SERIAL COMMUNICATION USING CONTROLLER AREA NETWORK PROTOCOL

2.1 AN INTRODUCTION TO COMMUNICATION PROTOCOLS

The factory floor is becoming an increasingly Internet connected and networked environment. The increasing demand for communication has led to the development of various communication protocols. These protocols differ from one another based on the application for which they have been designed. In particular, the communication protocols addressing industrial applications comprise the “higher end” and the “lower end” protocols. The higher end protocols, also called factory bus, address the overall factory system information, while the lower end protocols, called the fieldbus, address the inter processor communication as well as the sensor/actuator communication

The cost advantage and advanced functionality offered by silicon technology coupled with the growing dependence on distributed systems and networking, have resulted in the demand for newer means of highly standardized communication in the fieldbus application area. The industry requires flexible control systems characterized by a high degree of standardization. The high degree of standardization leads to reusable solutions in terms of hardware and software modules that are easy to adapt to the varying needs and solutions in each individual field of application. The number of fieldbus protocols has increased steadily over the last two decades, with Profibus, InterbusS, PNet, LON and FIP being among the earlier standards to be accepted.

The automobile industry developed several electronic systems to meet the growing demand for greater safety, comfort, convenience and compliance requirements for improved pollution control and reduced fuel consumption. As a result the complexity of these control systems and the need to exchange data among them required more and more hardwired, dedicated signal lines. This prompted the replacement of the existing mode of wiring by a network architecture where all the nodes in the network communicate through a common bus. The Controller Area Network protocol developed by R. Bosch GmbH, Germany is ideally suited for communication through the above network. Utilizing CAN, controllers, sensors, and actuators communicate with each other, in real time, at speed of up to 1Mbits/s, over a two wire serial data bus

The generic communication interface architecture essential to CAN protocol, models the Virtual Leveled Systems Architecture (VLSA) which is based on the concept of “Shared Variables”. In this model, the individual tasks reside in the distributed controllers, each being responsible for a part of the overall control program. An example of such a distributed system

is illustrated in Fig. 2.1.

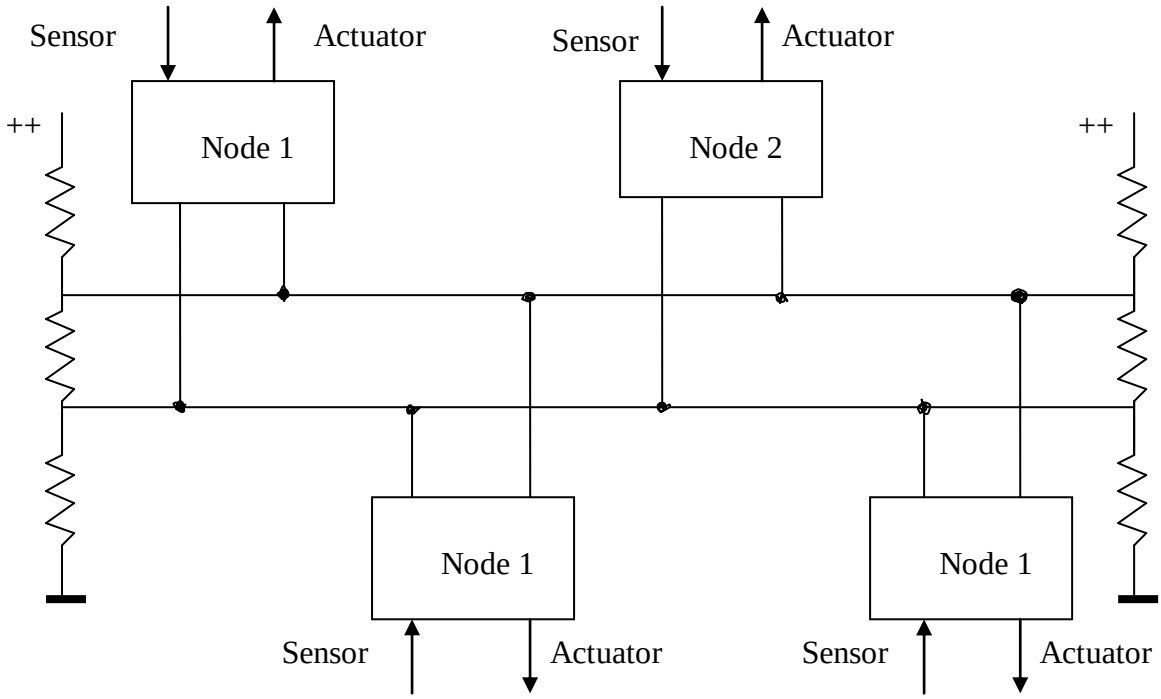


Fig 2.1 representation of a distributed network

In the above illustrated figure all the nodes are connected to a common CAN bus. The nodes interact with the actual process through their sensors and actuators. To transmit messages on the bus the nodes make use of a dynamic, priority based arbitration mechanism. Every node in the network will receive any message transmitted on the bus, the nodes filter out the relevant messages based on a message filtering algorithm. Based on the received message the application might send out control signals to the device through the actuators.

2.2 CAN PROTOCOL

The CAN protocol is an international standard defined in the ISO 11898 and ISO 11519. The difference between the two is in the physical layer, where ISO 11898 handles high speed applications up to 1Mbit/second. ISO 11519 has an upper limit of 125kbit/second. The CAN protocol only comprises of the Data Link composed of the Logical Link Control (LLC) sub layer and the Media Access Control (MAC) sub layer.

Data Link Layer	**LLC (Logic Link Layer)** Acceptance filtering Overload notification Recovery management **MAC (Medium Access Control)** Data encapsulation/decapsulation Stuffing/de-stuffing Bus arbitration Error detection Error signaling Fault confinement Acknowledgement
Physical Layer	**PLS (Physical Signaling)** Bit encoding/decoding Bit timing Synchronization **MDI (Medium Dependent Interface)**

Fig 2.2 CAN protocol ISO/OSI layered model

The CAN Data Link Layer controls the message communication. The Data Link Layer builds data frames to hold data and control information. It also provides other services such as frame identification, bus arbitration, bit stuffing, error detection, error signaling, fault confinement and automatic retransmission of erroneous frames.

The CAN Physical Layer is responsible for transfer of data between different nodes in a given network; it defines how signals are transmitted and therefore deals with issues like encoding, timing and synchronization of the bit stream to be transferred

The application layer is specified by higher layer protocols such as CAL/CANOpen and CAN Kingdom, DeviceNet.

The CAN ISO standard discussed in this thesis is specified by ISO 118981, which gives the data link layer and the physical signaling, and ISO 118982, which specifies the highspeed physical layer characteristics.

2.3 CAN DATA LINK LAYER

2.3.1 BUS ARBITRATION

CAN is a protocol for short messages. Each transmission can carry 0-8 bytes of data. It uses CSMA/CD+AMP (Carrier Sense Multiple Access/Collision Detection with Arbitration on Message Priority). Thus the protocol is message oriented and each message has a specific priority according to which it gains access to the bus in case of simultaneous transmission. An ongoing transmission is never interrupted. During the bus idle state, any node can access the CAN bus. In the event of multiple accesses, the priority is decided by a method called "NonDestructive Bitwise Arbitration". Nondestructive means that the winner of the arbitration the message with the higher priority must continue transmitting the message without having to restart the message transmission from the beginning

Arbitration is performed during the transmission of the identifier field. Each CAN message has an identifier which is of 11 bits. This identifier is the principle part of the CAN arbitration field, which is immediately after the Start bit. The identifier not only identifies the type of message but also indicates the priority of the message. During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node has lost arbitration. As soon as a node has lost the arbitration the node ceases to transmit, becoming a receiver of the ongoing message. At the end of transmission of the Arbitration Field, all nodes but one would have lost arbitration, and the highest priority message will get through unimpeded.

When the bus is free again the CAN Controller automatically makes a new attempt to transmit its message. Another round of arbitration is performed and the message with the highest priority gets through again leaving the messages with lower priority to content for the bus in the subsequent cycles. The CAN protocol requires that a specific identifier is sent only by one node this ensures that no two messages with the same identifier contend for bus access.

Fig. 2.3 illustrates the method of nondestructive bitwise arbitration of CAN. Two nodes A and B have a transmission request. The CAN bus bit indicates the bit value sampled from the bus and the value is the WiredAND value of the bits transmitted by nodes A and B.

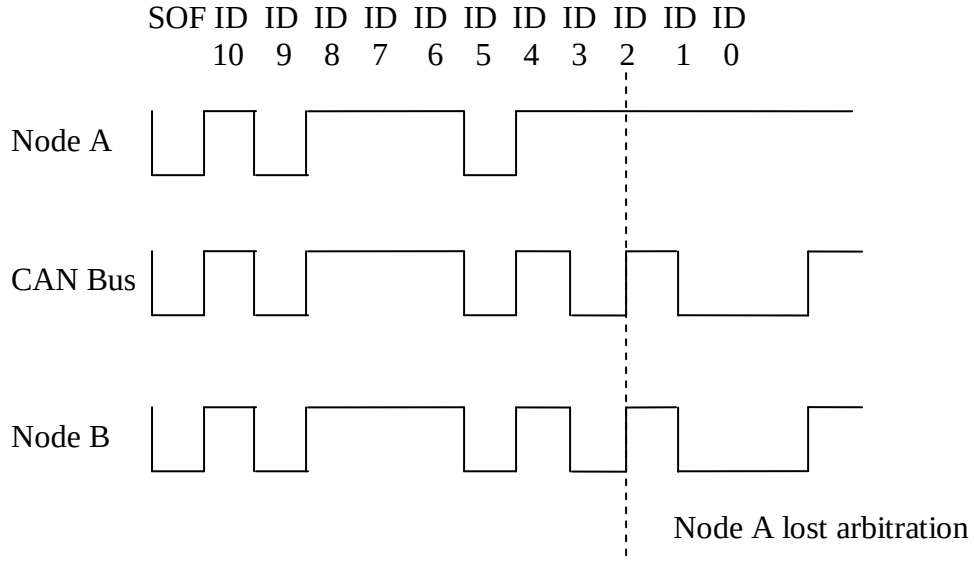


Fig 2.3 CAN bus arbitration

In the above example both nodes start transmission as soon as the bus is free. Both Node A and Node B transmits the same bits till the 4th bit of the Identifier frame and neither node loses arbitration. During the 4th bit time Node A transmits a dominant bit, Node B transmits a recessive level. The recessive bit level is overwritten by the dominant bit level. At this point of time Node A loses the arbitration. Node B stops the transmission of any further bits and switches to receive mode, because the message that has won the arbitration must possibly be processed by this node.

2.3.2 CAN MESSAGE TRANSFER

CAN uses short fixed format messages of different but limited length the maximum utility load is 94 bits. There is no explicit address in the messages; instead, the messages can be said to be content addressed, that is, their content implicitly determines their address. The CAN protocol specifies four different frame types for communication: Data Frame, to transfer data; Remote Frame, to request data; Error Frame, to globally signal a detected error condition; Overload Frame, to extend delay time between subsequent frames.

DATA FRAME

The Data Frames is the most common message type, CAN systems, uses Data Frames to transmit data over the network. A Data frame is of fixed format with varying but limited

data length. A Data Frame can hold up to eight bytes of data. A Data Frame is composed of eight different bit fields: Start of Frame; Arbitration Field; Control Field; Data Field; CRC field; Acknowledgement Field; End of Frame field and the Inter Frame Space.

The CAN protocol specifies two versions of the Frame, the Base Format and the Extended Format. The CAN specification 2.0A defines the Base Format CAN systems where the frames have standard 11 bit identifiers while the CAN specification 2.0B defines the Extended Format CAN systems where frames have 29 bit identifiers. The extended format is used in complex systems with heavy traffic where the number of messages created by transmitters on the network is greater than the number of possible ID codes that the CAN system could assign to them to make sure that each message is unique. The Standard CAN 11bit identifier provides for 2, or 2048 different message identifiers, while the Extended CAN 29bit identifier provides for 2, or 537 million identifiers. This design incorporates the CAN 2.0A specification and the following discussion will explain it in more detail.

1 Bit	11 Bits	1 Bit	1 Bit	1 Bit	4 Bits	0-8 Bytes	15 Bits	1 Bit	1 Bit	1 Bit	7 Bits	3 Bits
Start of Frame	Message Identifier	Remote transmission Request	Identifier Extension	Reserve bit r0	Data Length Code	Data Field	CRC sequence	CRC delimiter	Acknowledgement slot	Acknowledgement Delimiter	End of frame	Inter frame space

Fig 2.4 CAN data frame

The various fields of the data as shown in Fig. 2.4 are explained as follows: Start of Frame: A single dominant bit (logic 0) marks the start of the Data Frame. The CAN bus is in an idle (recessive) state prior to the transmission of this bit. All receivers on the bus use this

bit to synchronize their clocks to the transmitter's clock. Arbitration Field: The Arbitration Field is made up of 12 bits, the 11 bit Message Identifier and the Remote Transfer Request (RTR) bit. This field serves a dual purpose; it is used to indicate the logical address of the message and to determine which node has access to the CAN Bus.

The Message Identifier's bits are transferred in the order from ID10 to ID0 to satisfy the Nondestructive bitwise arbitration method of CAN. The lower the value of this field, the higher is the priority of the message. A message with all the bits of the identifier set to logic 0 will have priority over any other message in the network.

The RTR bit indicates whether the frame is a Data Frame or a Remote Frame. Within a Data Frame the RTR bit must be dominant. The RTR Bit is used by a receiver to request a remote transmitter to send its information. If this bit is set to 1 (= recessive) the frame contains no data field even if the data length code defines something other. A corresponding transmitter on receiving a Remote Transfer Request will initiate the transmission of a Data Frame. The request and the possible answer are two completely different frames on the bus. This means the response can be delayed due to messages with higher priorities

In the event of a data frame and a remote frame being transmitted simultaneously with identical message identifiers, the RTR bit decides the priority. As a data frame has a dominant RTR bit, it is given priority over a remote frame with a recessive RTR bit. Control Field: The control field is made up of 6 bits, the Identifier Extension (IE) bit, r0 and the 4 bit Data Length Code (DLC).

The first bit of the Control Filed is the IE; this bit is transmitted as a dominant bit in the standard format message to indicate that there are no more identifier bits in the message. The second bit is a reserved bit, transmitted as a dominant bit. The last four bits contain the data length code for the following data field. The DLC indicates the number of bytes in the Data Field. The Admissible values of the DLC field are zero to eight. Since the field is four bits long values greater than eight can be indicated. If the value of the Data Length Code is greater than eight then it is assumed that the frame contains eight bytes [10]. Data Field: This field holds the application data to be transferred. This field is of variable length. The Data Field can contain zero to eight bytes of data. The data is transmitted with the MSB first. CRC Field: The CRC Field is made up of the 15 bit CRC sequence and the recessive CRC Delimiter bit. The receiver uses the CRC sequence to check if the data bit sequence in the frame was corrupted during delivery. Acknowledgement Field: This field is of 2 bits and contains the Acknowledgement Slot bit and the recessive Acknowledgement Delimiter bit. A transmitter transmits a recessive Acknowledgement bit. Any receiver on successful reception

of the message sends out a dominant bit during this slot to acknowledge the successful reception of the message. A transmitter on reading back a dominant bit in the Acknowledgement Slot understands that at least one node if not all has received a complete and error free message. End of Frame Field: The EOF marks the end of the CAN Frame [10]. The End of Frame is made up of a sequence of seven recessive bits. Inter Frame Space: Every Data or Remote Frame is separated from the preceding frames by the Inter Frame Space. The IFS is made up of a sequence of recessive bits which extends for at least 3 bit durations. The bus may continue to remain idle after this, or a new frame will be indicated with the dominant Start of Frame bit. If one of the first two bits of this field is dominant then it is assumed an Overload Frame has been initiated.

REMOTE FRAME

In a CAN network a node acting as a receiver for certain information may initiate transfer of the respective data using a Remote Transmission Request message. Remote requests are sent out on a regular basis to get updates from other nodes. This feature is very useful in situations where a network node that was temporarily offline wishes to reconnect to the network. The node might have information that is not up to date. In such a case the node need not wait until the corresponding transmitters send messages. The node can update the information that was previously available to it by scanning all necessary messages after sending out Remote Frames to the corresponding transmitters on the network.

The receiver via the Message Identifier of the Remote Frame addresses the node from which it needs the data. Every node in the network receives the RTR frame. Based on the identifier the application detects if it is the corresponding transmitter of the message. A corresponding transmitter on receiving a Remote Transfer Request will initiate the transmission of a Data Frame. The request and the possible answer are two completely different frames on the bus. This means the answer can be delayed due to messages with higher priorities in addition to the possible delay by the application. An advantage of this feature is that the message by the transmitter containing the application data is not only received by the requesting receiver but also possible by other receivers which are interested in this message. This mechanism ensures the data consistency of the network.

1 Bit	11 Bits	1 Bit	1 Bit	1 Bit	4 Bits	15 Bits	1 Bit	1 Bit	1 Bit	7 Bits	3 Bits
Start of Frame	Message Identifier	Remote transmission Request	Identifier Extension	Reserve bit r0	Data Length Code	CRC sequence	CRC delimiter	Acknowledgement slot	Acknowledgement Delimiter	End of frame	Inter frame space

Fig 2.5 CAN Remote frame

The structure of the Remote Frame is similar to the Data Frame but for a few differences. Contrary to Data Frames, the RTR bit of Remote Frames is 'recessive'. There is no Data Field, independent of the values of the Data Length Code which may be signed any value within the admissible range 0 to 8 Bytes. The value is the Data Length Code of the corresponding Data Frame. The structure of a RTR messages can be seen in Fig. 2.5. The descriptions of the various fields in a Remote Frame are the same as in a Data Frame, but for the differences explained earlier.

ERROR FRAME

A CAN node on detecting an error, signals the presence of the error by sending out an Error Frame. The Error Frame can be sent during any point in a transmission and is always sent before a Data Frame or Remote Frame has completed successfully. The transmitter constantly monitors the bus while it is transmitting. When the transmitter detects the Error Frame, it aborts the current frame and prepares to resend the message once the bus becomes idle again. This ensures data consistency throughout the network.

The error frame consists of 2 different fields. The first field is the Error Flag given by

the superposition of Error Flags contributed by different CAN nodes. The second field is the Error Delimiter.

ERROR FLAG

Error Flags are made up of a sequence of six consecutive bits of the same polarity. This sequence of bits violates the rule of bit stuffing or destroys a bit field requiring fixed form. As a consequence all other nodes also detect an error condition and likewise start transmission of an error flag.

The Error Flag is of two types:

Active Error Flag: An Active Error flag is transmitted by an Error Active node. Active Error Flags are made up of a sequence of six consecutive dominant bits. This sequence violates the bit stuffing rule or destroys a bit field requiring fixed form. The total length of the Active Error Flag varies between a minimum of 6 and a maximum of 12 bits. This is due to the superposition of different error flags transmitted by individual nodes. Fig 2.6 gives the structure of Active Error Frame.

Passive Error Flag: A Passive Error Flag is transmitted by an Error Passive node. Passive Error Flags are made up of a sequence of six consecutive recessive bits. An Error Passive station waits for six consecutive bits of equal polarity, beginning at the start of the Passive Error Flag. The Passive Error Flag is complete when these six equal bits have been detected. Passive error flags initiated by a transmitter cause errors at the receivers when they start in a frame field which is encoded by the method of bit stuffing, because they then lead to stuff errors detected by the receivers. Passive error flags initiated by receivers are not able to prevail in any activity on the bus line. Fig. 2.7 gives the structure of a Passive Error Frame.

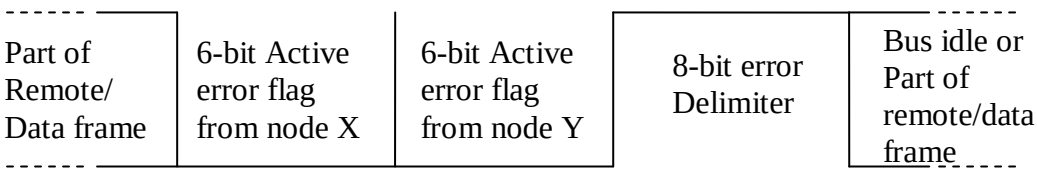


Fig 2.6 Active error frame

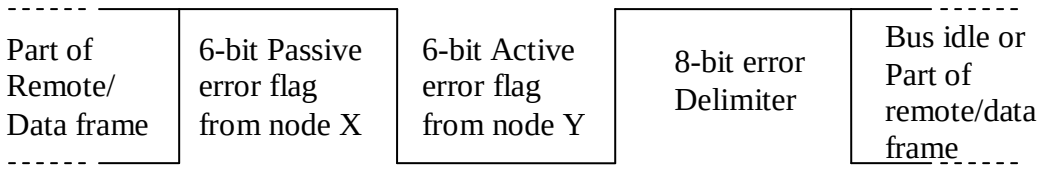


Fig 2.7 Passive error frame

Error Delimiter

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

OVERLOAD FRAME

Overload Frames are generated when a CAN controller encounters a situation in which it will not able to process a received message. The structure of the Overload frame is similar to that of an Active Error Flag. The only difference is that an Overload Frame is initiated at the last bit of the EOF field or in the IFS field. To delay the transmission of the next message, a CAN node sends out an Overload Frame, all of the nodes on the network detect it and send out their own Frames, effectively stopping all message traffic on the CAN system. Then, all of the nodes listen for a sequence of eight recessive bits before they contend for Bus access. The maximum amount of time needed to recover in a CAN system after an Overload Frame is 31 bit times. The structure of an Overload Frame is as shown in Fig. 2.8.

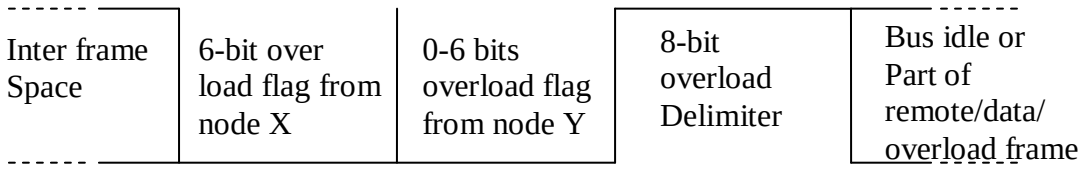


Fig 2.8 Overload frame

Acknowledgement

All receivers check for the consistency of the message being received. On receiving an error free message till the end of the CRC Sequence the CAN controller sends out a dominant bit in the Acknowledgement slot. A transmitter on reading back a dominant bit in the Acknowledgement Slot understands that at least one node if not all has received a complete and error free message.

INTERFRAME SPACE

Data frames and remote frames separated by Interframe space from the preceding frame of any type. In contrast, overload frames and error frames are not preceded by any interframe space. Interframe is different for error active and error passive nodes.

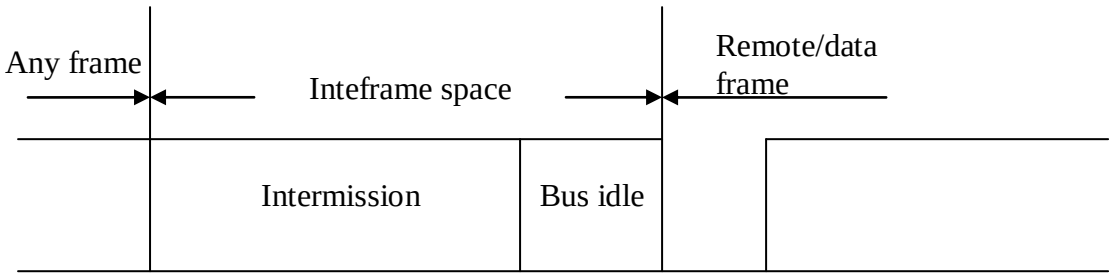


Fig 2.9 Interframe space for error active nodes

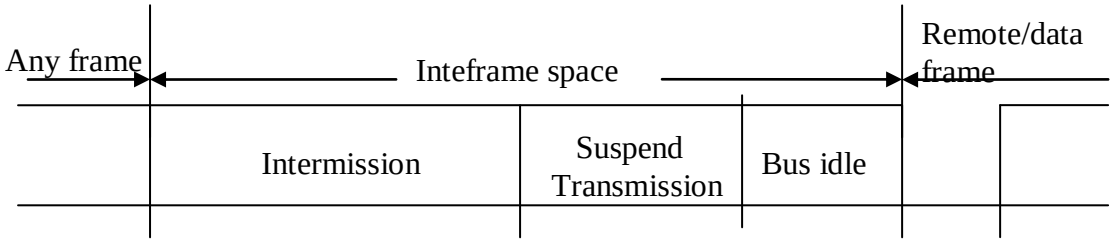


Fig 2.10 Interfarme space for error passive nodes

Intermission No nodes are allowed to start a transmission during intermission, which contains three recessive bits. The only action permitted is signaling of an overload condition.

Bus idle The bus idle is recognized as free bus, having arbitrary length. Any node can access

the bus for transmission and a dominant bit in the bus is treated as start of frame.

Suspend transmission An error passive node sends eight recessive bit after intermission, before starting to transmit a message or recognizing the bus to be idle. If meanwhile any other error active node stars transmission the node will become the receiver of the message.

2.3.3 BIT STUFFING

The CAN protocol specifies the use of NRZ encoding, this might cause the networks individual clocks to go out of sync. To enable synchronization even in the NRZ format, a bit of complementary polarity is inserted after every five consecutive bits of the same polarity in the bit stream to be transmitted. This practice is called bit stuffing.

The frame segments Start of Frame, Arbitration Field, Control Field and CRC sequence are coded by the method of bit stuffing. The remaining bit fields of the data frame or remote frame the CRC delimiter, ACK field and EOF are fixed form and are not stuffed. The Error Frame and the Overload Frame are of fixed form as well, and are not coded with bit stuffing

This method of bit stuffing is also significant keeping in mind the error signaling mechanism of the CAN protocol. The error frame transmitted to signal the presence of errors consists of an error of 6 consecutive bits of the same polarity. Since bit stuffing is used, six consecutive bits of the same polarity appearing in the received bit stream is considered an error. Whenever an error frame is transmitted a stuff error or a form error is detected in all the participating nodes causing the transmitter to abort transmission and retransmit the message once the bus is idle. The bit stuffing mechanism is demonstrated in Fig. 2.9.

Original Bit Stream to be transmitted – 010111111011

Stuffed Bit Stream transmitted on bus – 0101111101011

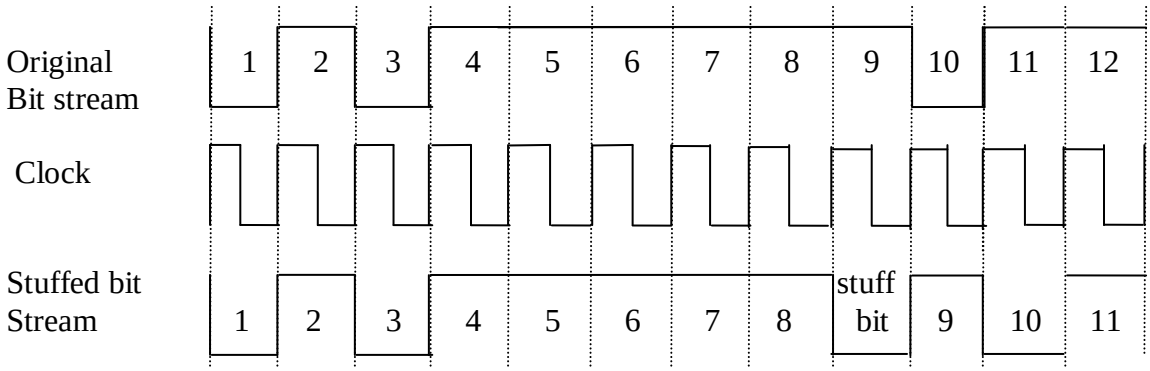


Fig 2.11 Bit stuffing

The receivers use the stuff bit to synchronize their clocks in case there is a long sequence of similar bits. During reception, the corresponding node automatically disregards the stuffed bit in the received bit stream to extract the information.

2.3.4 ERROR PROCESS

The robustness of CAN may be attributed in part to its abundant error checking procedures. Every error that is detected by a network node will be notified to the rest of the network immediately. Unlike other field bus protocols which apply the principle of message confirmation, in the CAN protocol only a corrupted message is signaled by means of an error frame.

The robustness of CAN may be attributed in part to its extensive and sophisticated error checking procedures. Every error that is detected by a network node will be notified to the rest of the network immediately. Unlike other field bus protocols, the CAN protocol does not apply the principle of message confirmation but instead signals errors immediately as they occur by means of an Error Frame. There are several mechanisms in the CAN protocol, to detect errors and to prevent erroneous nodes from disabling all bus activities.

The CAN error process is divided into three parts: Error Detection, Error Handling, and Error Confinement.

ERROR DETECTION

The CAN protocol implements five mechanisms for error detection; three at the message level, two at the bit level.

Message level:

Cyclic Redundancy Check (CRC): The CRC Field of every message holds the checksum of the preceding application data. This CRC sequence is transmitted in the CRC Field of the CAN frame. The receiving node performs a similar checksum of the received application data and performs a comparison to the received sequence. If the receiver detects a mismatch between the calculated and the received CRC sequence, then a CRC error has occurred. The receiver discards the message and transmits an Error Frame to request retransmission of the frame.

Form Check: There are certain predefined bit values that must be transmitted recessive within a message. If a receiver detects a dominant bit in one of these positions a Form Error

will be flagged. The bits checked are the CRC delimiter, ACK delimiter, and the EOF bits.

Acknowledgment Check: Every node that transmits a message listens for an acknowledgement in the ACK slot. An Acknowledgement Error is flagged if a transmitter does not monitor a dominant bit during the ACK.

Bit level:

Bit Monitoring: Every transmitting node monitors the transmitted bit level. If the monitored bit level is different from the transmitted bit level, a Bit Error is flagged.

Bit Stuff Monitoring: The bit stuffing rule specifies that a bit of opposite polarity is inserted after every five consecutive bits of the same polarity. If any receiving node detects six consecutive bits of the same polarity between Start of Frame and the CRC Delimiter, the bit stuffing rule has been violated. A stuff error occurs and an Error Frame is generated. The message is then resend.

ERROR HANDLING

A CAN node on detecting an error condition, using any and all of the five mechanisms described above, signals the presence of an error by transmitting an Error Frame. The Error Frame transmitted by the node that detects the error, induces a Stuff or Form Error in other receiving nodes and a Bit Error in the transmitting node. This causes the transmitting node to abort transmission of the message. Since all the participating nodes receive the Error Frame the erroneous message is discarded thus ensuring the consistency of data throughout the network. The transmitter of the message will retransmit the message when the bus is free again. The retransmitted message still has to contend for arbitration on the bus.

ERROR CONFINEMENT

If a message fails an error frame is generated from the receiving nodes, causing the transmitting node to resend the message until it is received correctly. The probability of a network breakdown because of a local disturbance of one or a group of network nodes is very high. The CAN protocol makes use of a unique algorithm to prevent such a scenario. The algorithm is designed to automatically detect a faulty node and disconnect it from the network by removing the nodes transmit capability on reaching an error limit.

To implement the Error Confinement mechanism, CAN makes use of two error counters, one to keep track of transmit errors (Transmit Error Counter) and the other to keep

track of receive errors (Receive Error Counter). A non proportional method of counting is implemented in CAN. The counter increment value in case of a transmission or reception error is higher than the counter decrement value for every successful transmission or reception. The state diagram for Error Confinement is given in Fig. 2.10.

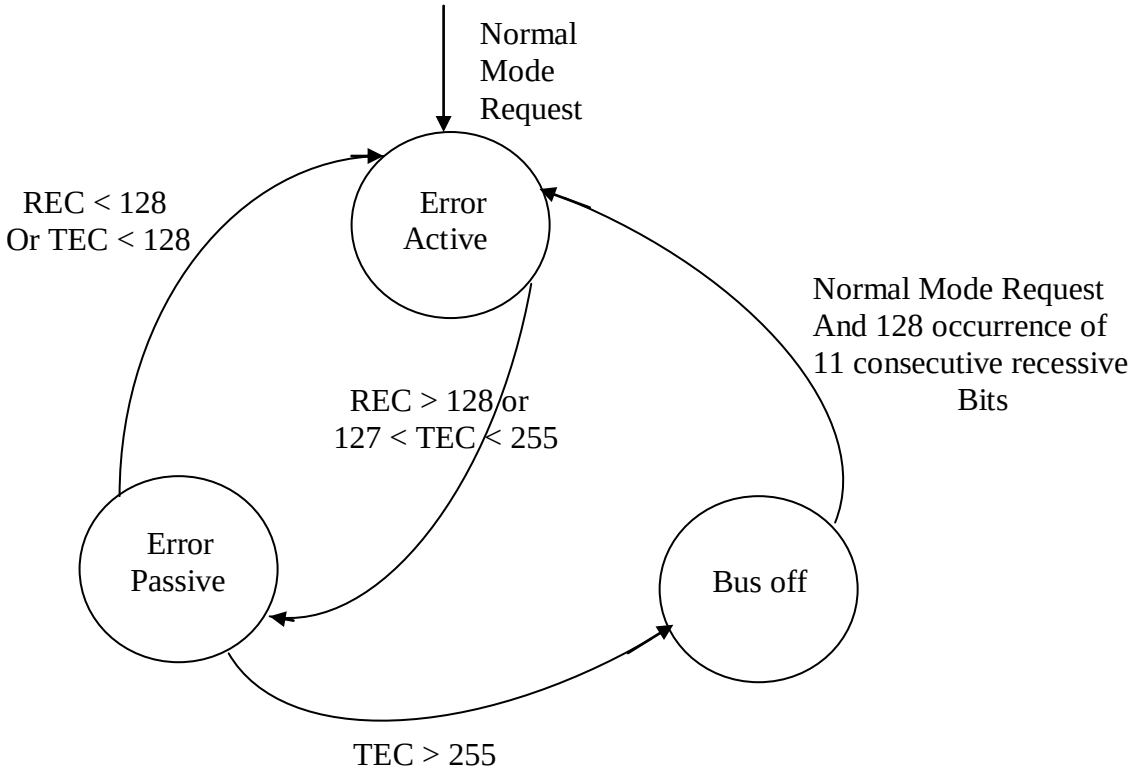


Fig 2.12 CAN error states

Based on these counters, a node can be in one of three states:

Error Active: This is the normal operating state of the CAN node. In this state the node can transmit and receive messages. In the event of an error the node transmits an Active Error Frame. The node is in Error Active state if the values of both the REC and TEC are less than or equal to 127.

Error Passive: In this state the node can take part in normal bus communication. The node transmits a Passive Error Frame to signal the presence of errors. A transmitting node in Error Passive state has to wait for an extra 8 bit bus idle period following the Inter Frame Space before it contends for Bus access. A node enters the Error Passive state when one of its counter values exceeds 127. The node can still return to the Error Active state when both the counters value is less than 128.

Bus Off: In the Bus Off state the nodes participation is restricted to receiving messages. The

node can no longer transmit messages. A node enters the Bus Off state if the Transmit Error Count of a node exceeds 255. No messages can be transmitted until the error counters are reset by the host microcontroller or processor.

1. The Error Counts are modified as per the rules specified in the BOSCH, CAN Specification Version 2.0. More than one rule may apply during a given message transfer
2. When a receiver detects an error, the REC will be incremented by 1, except when the detected error is a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
3. A receiver on detecting a dominant bit as the first bit after sending an Error Flag increments the REC by 8.
4. When a transmitter sends an Error Flag the TEC is incremented by 8. The TEC remains unchanged, under the following exceptions:
5. Exception 1: If an Error Passive transmitter detects an Acknowledgement Error because of not detecting an Acknowledgement and does not detect a dominant bit while sending its Passive Error Flag.
6. Exception 2: If the transmitter sends an error flag because a stuff error occurred during arbitration whereby the stuffbit is located before the RTR bit, and should have been recessive, and has been send as recessive but monitored as dominant.
7. The TEC is incremented by 8 if a transmitter detects a Bit Error while sending an Active Error Flag or an Overload Flag.
8. The REC is incremented by 8 if a receiver detects a Bit Error while sending an Active Error Flag or an Overload Flag.
9. A node can tolerate up to 7 consecutive dominant bits after transmitting an Error Flag or Overload Flag. Every Transmitter increments its TEC by 8 and every receiver increments its REC by 8, after detecting the 14th consecutive dominant bit (in case of an Active Error Flag or an Overload Flag) or after detecting the 8th consecutive dominant bit following a Passive Error Flag, and after each sequence of additional eight consecutive dominant bits.
10. The TEC is decremented by 1 (unless it is already 0) after the successful transmission of a message.
11. After the successful reception of a message and the successful transmission of the Acknowledgement, the REC is decremented by 1, if its value is between 1 and 127. If the REC is 0, it stays 0, and if it is greater than 127 it is set to a value between 119

and 127.

12. If the TEC or REC equals or exceeds 128 the node will change to error passive state.
13. If TEC is equal or greater than 256 the node will become in bus off state.
14. An error passive node again return to error active state when both the TEC and REC return to less than or equal 127.
15. A node which is bus off is permitted to become error active (no longer) bus off with its error counters both set to 0 after 128 occurrences of 11 consecutive recessive bits have been monitored on the bus.

2.3.5 ACCEPTANCE FILTERING AND MESSAGE VALIDATION

Data messages transmitted from any node on a CAN bus do not contain addresses of either the transmitting node, or of any intended receiving node. Instead, the content of the message is labeled by an identifier that is unique throughout the network. The content of the Arbitration Field is commonly used as a message identifier. All other nodes on the network receive the message and each performs an acceptance test on the identifier to determine if the message, and thus its content, is relevant to that particular node. The Acceptance Filter, based on the mask and code parameters of the host application, filters the received messages and stores only those required by the host application. Thus the host application ensures only relevant messages are processed and the rest are ignored.

OVERLOAD NOTIFICATION

A standard CAN controller has two Receive Message Buffers to store the received data (The Motorola MSCAN has four receive buffer). This enables the host application to process one message while it is receiving another message. In case both the buffers are full, the controller will not accept any further messages until one of the buffers is cleared by the host microcontroller. To ensure that none of the messages transmitted during this period is lost, the transmission of the next message is delayed by sending out an Overload Frame. All the nodes on the network detect the Overload Frame and send out their own frame, effectively stopping all message traffic on the CAN system. Then, all of the nodes listen for a sequence of eight recessive bits before they contend for Bus access. The delay produced by the Overload Frame is normally sufficient for the host application to clear one of the buffers. A maximum of two Overload Frames can be sent in succession to delay the transmission of a message on the Bus. An Overload Frame is initiated only at the last bit of the EOF field or in the IFS field

2.4 CAN PHYSICAL LAYER

The CAN Physical Layer is responsible for the physical connection between two nodes in a network and the actual transmission of electrical impulses. The CAN Physical Layer defines the bit representation, bit encoding, bit timing and synchronization and the bit rate and bus length scheme on the CAN bus.

The CAN Physical Layer translates the data provided by the transmitter's Data Link layer into an electronic signal. On the receiving end, the Physical Layer translates the electronic signals back into a data format and passes it on to the Data Link layer. There are several different physical layers, the most common type of is the one defined by the CAN standard, part ISO 118982. It is also referred to as "highspeed CAN". A simplistic representation of the CAN wiring diagram is shown in Fig. 2.13.

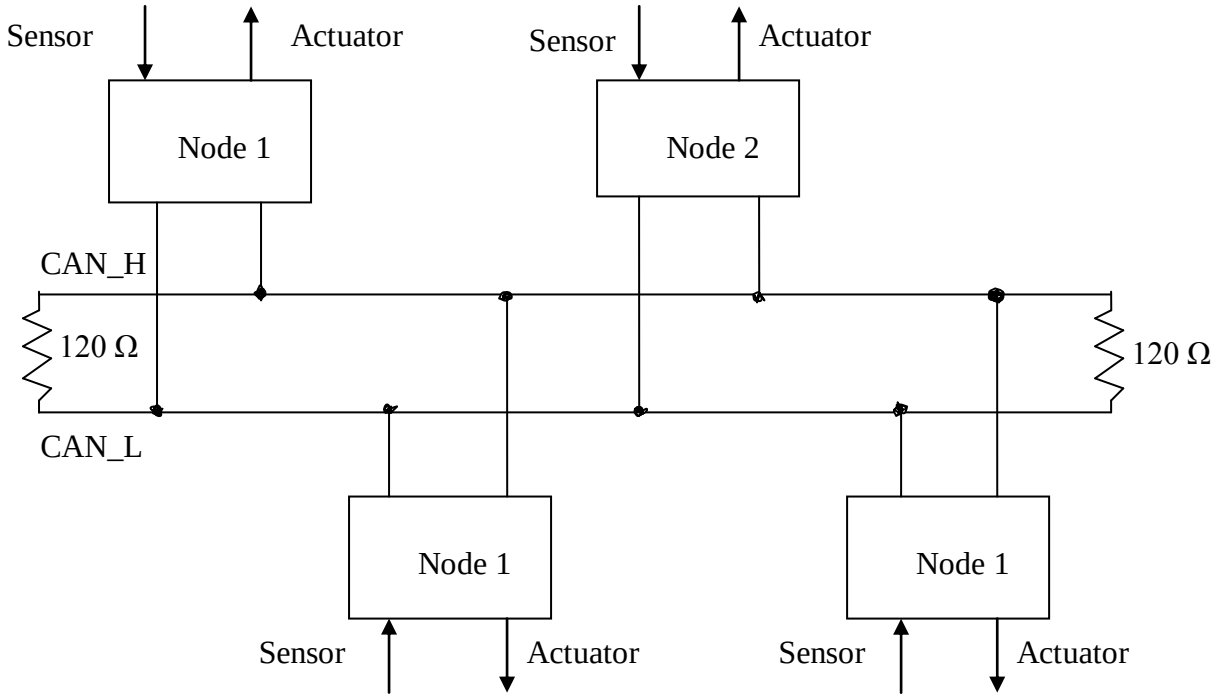


Fig 2.13 CAN network

Highspeed CAN is made up of two wires arranged as a differential pair. The two wires are named CAN_H and CAN_L. Since the wires operate in the differential mode they carry inverted voltages, this provides excellent immunity from external electrical interference. The Bus termination is done using a resistor of 120 Ohms at each end of the bus. The termination removes the signal reflections at the end of the bus and ensures the bus gets correct DC levels.

2.4.1 BIT REPRESENTATION

There are two logical bit representations used in the CAN protocol. CAN signals use the terms recessive and dominant to describe the state of the bus. The CAN protocol defines logic ‘0’ as a dominant bus state and logic ‘1’ as a recessive bus state. These correspond to certain electrical levels which depend on the Physical Layer used.

The modules are connected to the CAN bus in a wiredAND fashion. A recessive bit appears on the CAN Bus only if all the nodes on the network transmit a recessive bit during that bittime. Since dominant bits always overwrite recessive bits, even if one node were to transmit a dominant bit during that bit time, then the whole bus is in the dominant state regardless of the number of nodes transmitting a recessive state. The concept of dominant and recessive bus states is an important concept in the discussion of CAN Bus Arbitration.

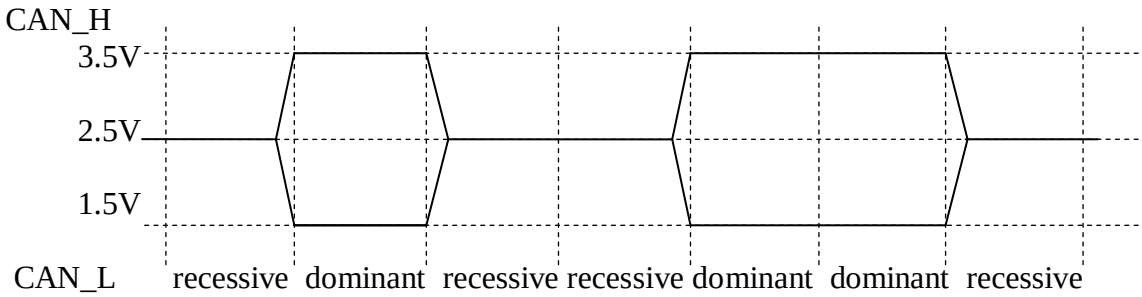


Fig 2.14 CAN dominant and recessive states

The bit representation followed in CAN is shown in Fig. 2.14. For a two wire High-speed CAN Bus the recessive bus state “logic 1”occurs when the CAN_L and CAN_H lines are at the same potential (CAN_L = CAN_H = 2.5V), and the dominant bus state “logic 0” occurs when there is a difference in potential (CAN_L = 1.5V and CAN_H = 3.5V). The voltage level on the CAN bus is recessive when the bus is idle.

2.4.2 BIT ENCODING

The CAN protocol specifies the use of Non Return to Zero (NRZ) encoding. In the Nonreturn to zero encoding the logic level of the bit remains the same throughout the bit duration. If a frame has a string of ‘1’s or ‘0’s, the signal will remain constant for as many bit times as necessary. The disadvantage of NRZ is that there is no easy way to tell where each

bit starts or ends when there are two or more ‘1’s or ‘0’s in a row. The only way to know where a bit starts or ends is for the receiver to have a clocking source that is identical to the transmitter so that it can decipher the bit stream. To enable clock synchronization among nodes in a CAN network, CAN uses the method of bitstuffing.

2.4.3 BIT TIMING AND SYCHRONIZATION

The Controller Area Network protocol uses synchronous data transmission to make the data transmissions more efficient. A major drawback of the synchronous transmission system is the absence of regular reference points for the node to perform bit synchronization. In a synchronous transmission system only one reference point is available at the start of the message. As it is difficult to keep any two clocks synchronized over a long period of time without some sort of reference point, CAN makes use of a sophisticated method of bit synchronization to overcome this drawback. In this method of synchronization every node in a network is continuously resynchronized. This ensures that all the nodes in a network are synchronized to function at the same transmission rate.

Bit rate is given by the number of bits that pass a given point in a network per second. The nominal bit rate is the number of bits per second transmitted by an ideal transmitter with no resynchronization. Whether transmitting or receiving, all nodes on the network must have the same nominal bit rate. The nominal bit time is given by the inverse of the nominal bit rate. The nominal bit time is the amount of time needed to transmit a single bit across the network.

CAN systems use the bit time to make sure that the nodes sample the bus at the appropriate time to determine whether the bus is in a recessive or dominant state. To accomplish this, the nominal bit time is broken into segments of four nonoverlapping time segments the synchronization segment, the propagation segment and the phase segments one and two. Each segment consists of one or more time quanta. The partition of the bit time is shown in Fig. 2.15.

The Synchronization Segment (Sync_Seg) is that part of the bit time where the signal transition is expected to occur. The Synchronization Segment is always fixed at one time quanta.

Propagation Segment (Prop_Seg) is that part of the bit time intended to compensate for the physical delay in the bus lines. The Propagation Segment size should at least be equal to

twice the time required for a bit to reach the farthest node. This is programmed to be 18 time quanta long.

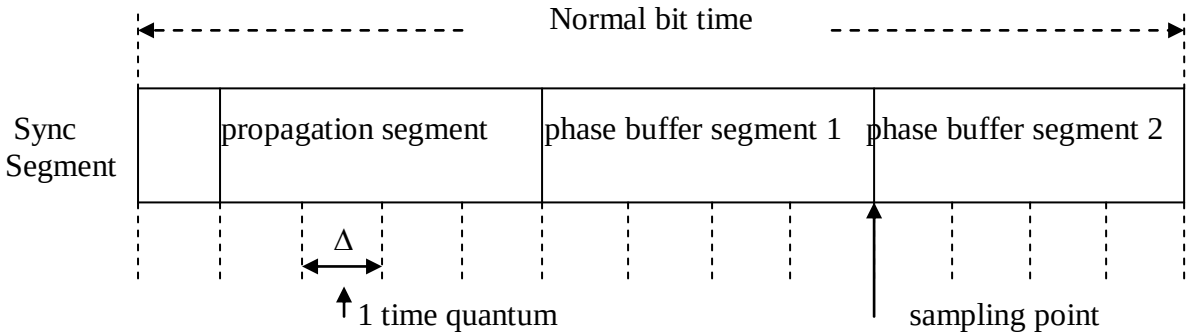


Fig 2.15 CAN bit time

Phase segment one (Ph_Seg1) is a buffer segment that is lengthened if a recessive to dominant transition occurs during the Propagation Segment. The Phase segment one is lengthened such that the distance from the edge to the sampling point is the same as it would have been from the Synchronous Segment to the sample point if no edge had occurred. This is programmed to be 1 to 8 time quanta long.

Phase segment two (Ph_Seg2) is also a buffer segment that is shortened if a recessive to dominant transition occurs during the Phase segment two. The Phase segment two is shortened such that the distance from the edge to the sampling point is the same as it would have been from the Synchronous Segment to the sample point if no edge had occurred. This is programmed to be the maximum of the Phase Buffer Segment 1 and the Information Processing Time.

The Information processing time is the time segment from the sampling point reserved for sampling the next bit level and is less than or equal to 2 time quanta.The Sampling Point is always at the end of Ph_Seg1 and is the time at which the bus level is read and interpreted as the value of the current bit. CAN uses two types of synchronization, Hard Synchronization and Resynchronization.

Hard synchronization occurs only once during a message transmission, on the recessive to dominant transition of the Start of Frame bit. The bit time is restarted at the end of the synchronization segment after a hard synchronization.

Resynchronization is subsequently performed during the remainder of the message frame, when a recessive to dominant transition occurs outside of the expected synchronization

segment.

The distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The resynchronization Jump Width (SJW) defines how far a resynchronization may move the Sample Point to compensate for edge phase errors . The maximum change that can be effected on the Phase Buffer Segments is given by the Min [magnitude of phase error, SJW].

2.4.4 BIT RATE AND BIT LENGTH

CAN belongs to the class of "InBitResponse Protocols**",** where a wave traveling from a node at one end of the network to a node at the other end of the network should not consume more than approx. 2/3 of a bit time. This constraint implies that the higher the bus length the lower the maximum allowed bit rate becomes.

CHAPTER 3

MOTOROLA MC9S12DP256B

3.1 INTRODUCTION

Automotive electronics has emerged as one of the most exciting and dynamic manufacturing sectors in the world. The amount of electronics incorporated into the modern car of this era is fastidious nowadays. Global industry analysts forecasting that by 2010 electronic part of the car will account for 40% of the cost of the typical midsized car. Car makers are competing with each other by offering latest electronic systems in body electronics, chassis, driver assistance, infotainment, in-vehicle networking, safety, etc. Freescale (Motorola) is a major supplier of sensors, analogue products and 8-, 16-, and 32-bit MCU families for the automotive electronics. Freescale was the first supplier to integrate CAN, LIN, and flash memory technologies on automotive MCUs. Their 16-bit MCUs are mainly focusing on central body electronics, chassis/safety, and powertrain control in automotive field.

3.2 MC9S12DP256B

MC9S12DP256B is a 16-bit microcontroller provided by Freescale consisting a 16-bit central processing unit (STAR 12 CPU), 256K bytes of flash EEPROM, 12K bytes of RAM, 4Kbytes of EEPROM, an Enhanced capture timer, an 8 channel pulse width modulator (PWM), two 8 channel, 10 bit analog to digital converters (ADC), two asynchronous serial communications interfaces (SCI), three serial peripheral interfaces (SPI), Five CAN 2.0 A/B software compatible modules (MSCAN 12), and an inter IC bus.

3.3 FEATURES

- 16 bit STAR 12 CPU

  It has a high speed 16 bit processing unit, and wider internal registers (up to 20 bits) for extended math instruction. It offers an extensive set of indexed addressing capabilities.

- Multiplexed external bus.
- Memory

  It consists of 256K bytes of flash EEPROM, 4Kbytes of EEPROM, and 12Kbytes of RAM.
- Two 8 channel analog to digital converters, 8/10 bit resolution.

- 8 channel enhanced capture timer

  Each channel can configure as input capture, output compare, or pulse accumulator.
- 8 PWM channels with programmable period and duty cycle.
- Five CAN 2.0A/B software compatible modules having a maximum baud rate of 1Mbps.
- Inter IC bus (I2C).
- Serial interfaces

  Two asynchronous serial communications interface (SCI)

  Three synchronous peripheral interfaces (SPI).
- Byte Data Link Controller (BDLC).
- System Integration Module (SIM)

  Includes Clock and Reset Generation (CRG) module, Multiplexed External Bus Interface (MEBI), Module Maping Control (MMC), Interrupt control, Background debug mode (BDM)
- Available in two packages, 112-pin LQFP package or 80-pin QFP package.
- Maximum speed of 50MHz equivalent to 25MHz bus speed.
- Generate 2.25 to 2.75V Digital supply voltage using an internal voltage regulator.
- Analog and I/O supply voltage 4.75 to 5.25V.
- 0.25 micron CMOS technology.

3.4 CPU (STAR 12)

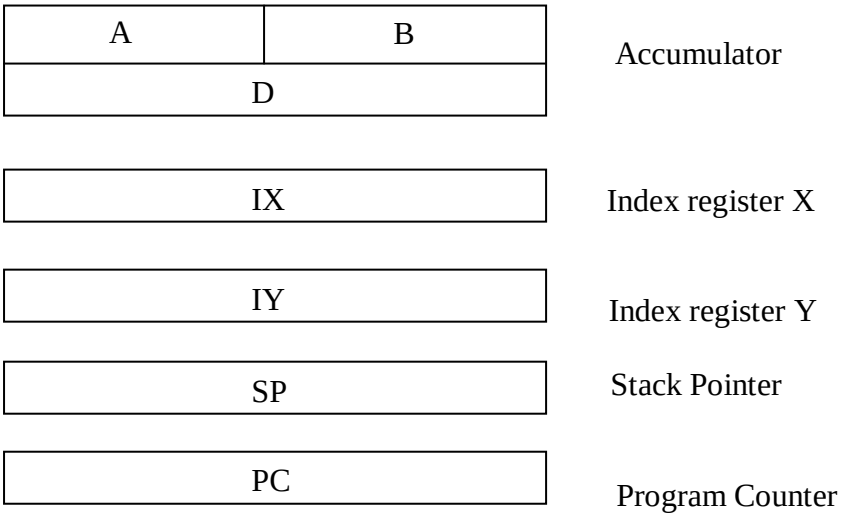
The STAR 12 CPU is a high speed 16 bit processor having a full 16 bit data paths and wider internal registers (up to 20 bits) for high speed extended math instruction. It has an instruction pipe to increase the execution speed. It supports an extensive set of indexed addressing mode.

Accumulator: A and B registers are the 8 bit general purpose accumulators used by the CPU to hold the operands and results for its processing. In 16 bit operation this will be treated as accumulator D.

Index registers: IX and IY are the index registers for keeping effective address in indexed addressing mode.

Stack pointer: Stack pointer points to the last location of the stack used. It can also be used as a pointer in indexed addressing mode. CPU supports an automatic program which stores the system contexts during subroutine call and interrupts.

Program counter: This 16 bit register points to the next instruction to be executed. It can be used in all indexed addressing mode except auto increment/ decrement.



S	X	H	I	N	Z	V	C

Condition Code Register

Fig 3.1 Register set for CPU 12

Condition code register: There are five status flags in the conditional code register, they are carry/ borrow flag (C), overflow flag (V), zero flag (Z), negative flag (N), and half carry flag (H). This register also contains two interrupt masking bits, X and I. S bit corresponds to stop disable bit.

The data types supported by this CPU are;

Bit data

8 bit and 16 bit signed and unsigned integers

16 bit unsigned fractions

16 bit address.

Addressing modes supported by the CPU are;

Direct addressing mode

Immediate addressing mode

Inherent addressing mode

Extended addressing mode

Relative addressing mode

Indexed addressing mode

The indexed addressing mode includes, 5 bit offset indexed addressing mode, auto pre-decrement, auto pre-increment, auto post-decrement, auto post-increment, accumulator offset, 9 bit offset, indexed 16 bit offset, indexed indirect 16 bit offset, and Indexed indirect accumulator D offset.

3.5 OPERATING MODES

Input BKGD & Bit MODC	MODB	MODA	Mode description
0	0	0	Special single chip, BDM allowed and active. BDM allowed in all other modes but a serial command is required to activate
0	0	1	Emulation expanded narrow, BDM allowed
0	1	0	Special test (Expanded wide), BDM allowed
0	1	1	Emulation Expanded wide, BDM allowed
1	0	0	Normal single chip, BDM allowed
1	0	1	Normal expanded narrow, BDM allowed
1	1	0	Peripheral; BDM allowed but bus operations would cause conflicts must not be used.
1	1	1	Normal expanded wide, BDM allowed

Table 3.1 Operating modes of STAR 12 CPU

The operating mode in which the microcontroller works, after reset determined by the MODA, MODB, and MODC pin status during reset. The register MODE which contains MODA, MODB, MODC bits, shows the current operating mode during operation.

3.6 SOME ON-CHIP RESOURCES

3.6.1 ENHANCED CAPTURE TIMER

Timers are the on-chip resource in all the real-time embedded systems. This microcontroller provides enhanced features in its timer module from the standard HC12 timer module to enlarge the application field, e.g. ABS in automotive field. The purpose of the timer module is to generate timed signals, to synchronize the program with timing events and

to capture external time events. . The timer module in this MCU contains a 16 bit free running up counter, input captures, output compares, pulse accumulators, and a 16 bit modulus down counter. It has 8 channels; each can be configured as input capture or output compare. There are four different clock sources for the timer module counter: timer prescaler clock, pulse accumulator clock (PACLK), PACLK/256, or PACLK/65536, which allows the timing signal duration varies from microseconds to many seconds.

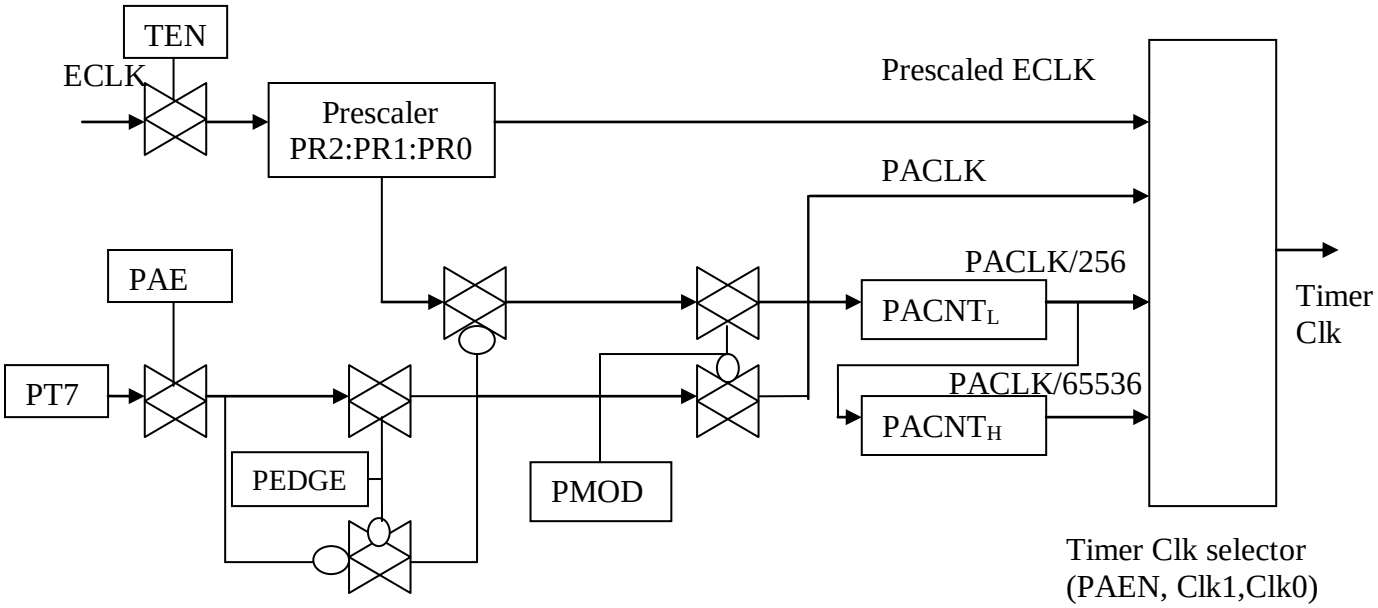


Fig 3.2 Timer module clock selection circuit and register

Input capture function allows capturing and holding of the free running counter value when an input event occurs. The timer can set to capture rising edge, falling edge or any edge of the input signal. The applications of input capture include event detection, event counting, pulse width measurement and frequency measurement.

Output compares generates an output action or timer event when the free running counter value matches with the count register. The different output actions on the timer channel are, clear, set, or toggle the respective channel output. This can be used to generate timing signals.

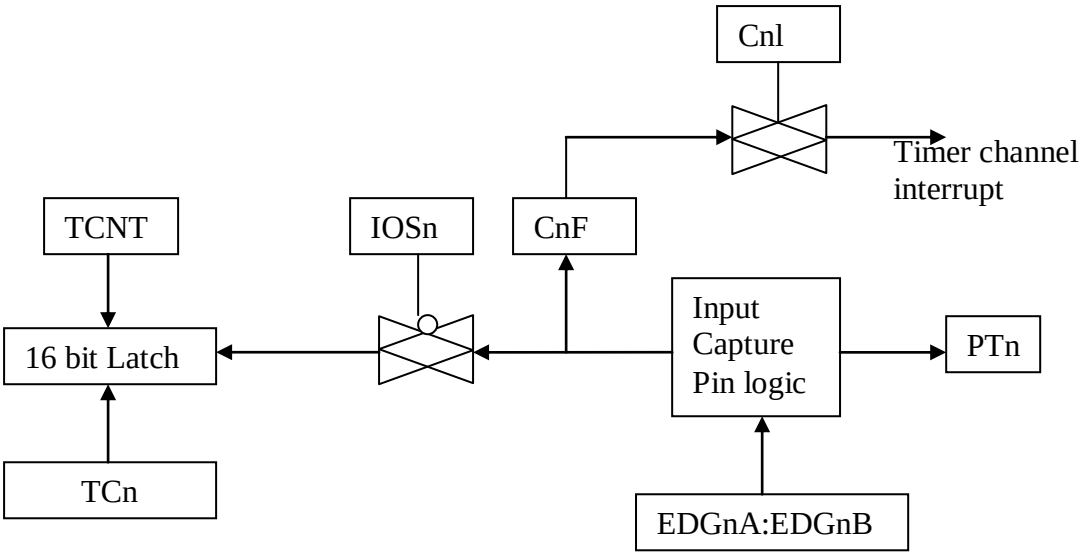


Fig 3.3 Input capture functional block diagram

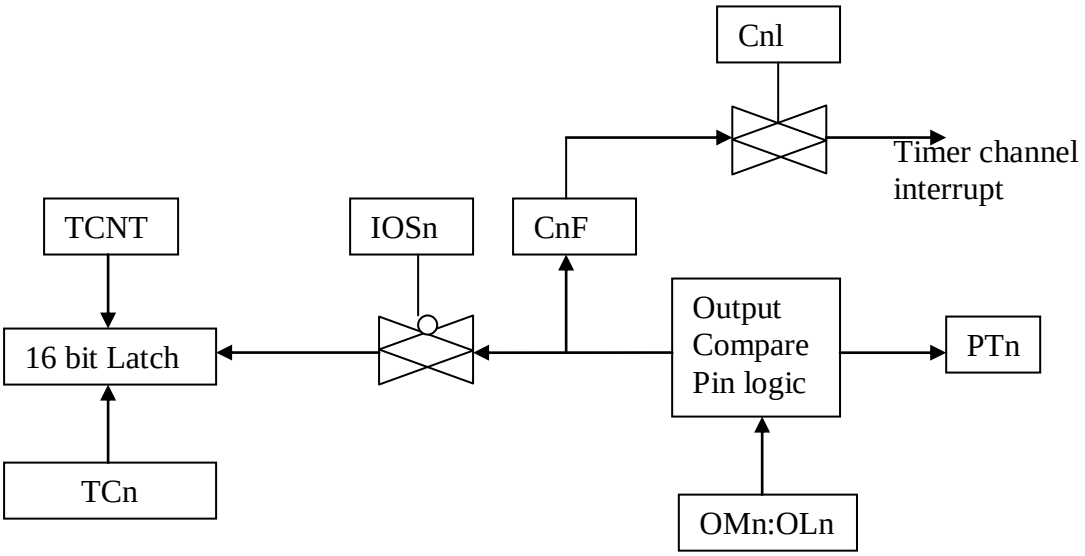


Fig 3.4 Basic output compare functional diagram

There are two pulse accumulators in the timer module, pulse accumulator A and pulse accumulator B. Both have a 16 bit counter and can be used in event counter mode or gate time accumulation mode. In event counting mode the 16 bit counter will increment for each event (rising or falling) on the input pin. In gate time accumulation mode the counter starts to count its clock whenever any gating signal occurs on the pin.

3.6.2 ANALOG TO DIGITAL CONVERTER

Since most of the embedded system inputs and outputs are analog, there should be modules to convert analog signals to digital and digital signals back to analog. The ATD (analog to digital converter) module composed in this microcontroller works based on successive approximation architecture having 8/10 bit resolution. The functionality is divided into three sub module, custom analog, the conversion mode control/register file, and the IP bus interface. Custom analog performs analog to digital conversions. All the control, status, test and result registers are located in the conversion mode control/register sub module. IP bus interface controls the communication between the entire module and CPU bus.

The main features of this module are,

It can provide 8/10 bit resolution

The minimum conversion time (for 10 bit single conversion) is 7µs

Programmable sample time

Result data in signed/unsigned, left/right justified manner

1 to 8 bit conversion sequence length

Analog input multiplexer for 8 analog input channels

Continues conversion mode

Input pin can configure as analog or digital input.

CUSTUM ANALOG

Sample and hold submodule:

The sample and hold (S/H) circuitry accepts the analog signals from outside world and stores it as capacitor charge on a storage node. During sampling process this module uses a sample amplifier to quickly charge the capacitor node according to the input voltage. When

not sampling, the sample and hold submodule disables its own clocks at its input. The input signal coming is unipolar and must be in the range specified.

Analog input multiplexer:

This multiplexer selects one of the 8 analog input channels to sample the analog input, and it can be extended to 16 analog channels. This contains an anti cross talk circuitry to avoid cross talk between the channels.

Sample buffer amplifier:

The sample buffer amplifier is used to buffer the input analog signals so that the storage node can quickly charge to its input potential.

Analog-to-digital converter submodule:

This submodule performs the conversion process using successive approximation method. It will compare the input signals with a series of digitally generated analog signals. By following a binary search algorithm the converter locates the approximate potential near to the sampled potential. When not sampling, the analog to digital submodule disables its own clocks at the input to the submodule.

Register and capacitor DAC arrays:

The heart of this analog-to-digital converter is a register capacitor array which performs the digital-to-analog conversion of the digital value comes as output for the binary search algorithm to compare with the input analog potential. The design of the 10 bit array is in pseudo differential scheme.

Comparator:

This module uses a three stage comparator. The first two stages are capacitive coupled differential comparator stage; these add a fixed gain to the comparison signal. The last stage is a clocked regenerative comparator stage; this stage drives the input comparison signal to the operating potential rails.

CONVERSION MODE CONTROL AND REGISTER FILES

This section is used by the programmer to interface with the ATD module. There are eight control registers and three status registers associated with this ATD module. The mode control communicates with the S/H machine and the A/D machine when necessary to collect samples and perform conversions.

LIMITATIONS:

The sample and hold module can hold one sample at a time. The conversion time depends on the settling time of the digitally generated compare signals. Here the settling time depends on the RC time constant of the resistor capacitor array, and it has a time constant approximately 1µs.

3.6.3 REAL TIME INTERRUPT

Real time interrupt (RTI) is a feature of clock and reset generation module, which generates periodic timer events. Any program segment can be synchronized to this event by either polling RTI event flag (RTIF) or by using the RTI interrupt. The RTI timer runs with the Oscillator clock. If this enabled, the interrupt will occur at the rate selected by the RTR bits in the RTICTL register.

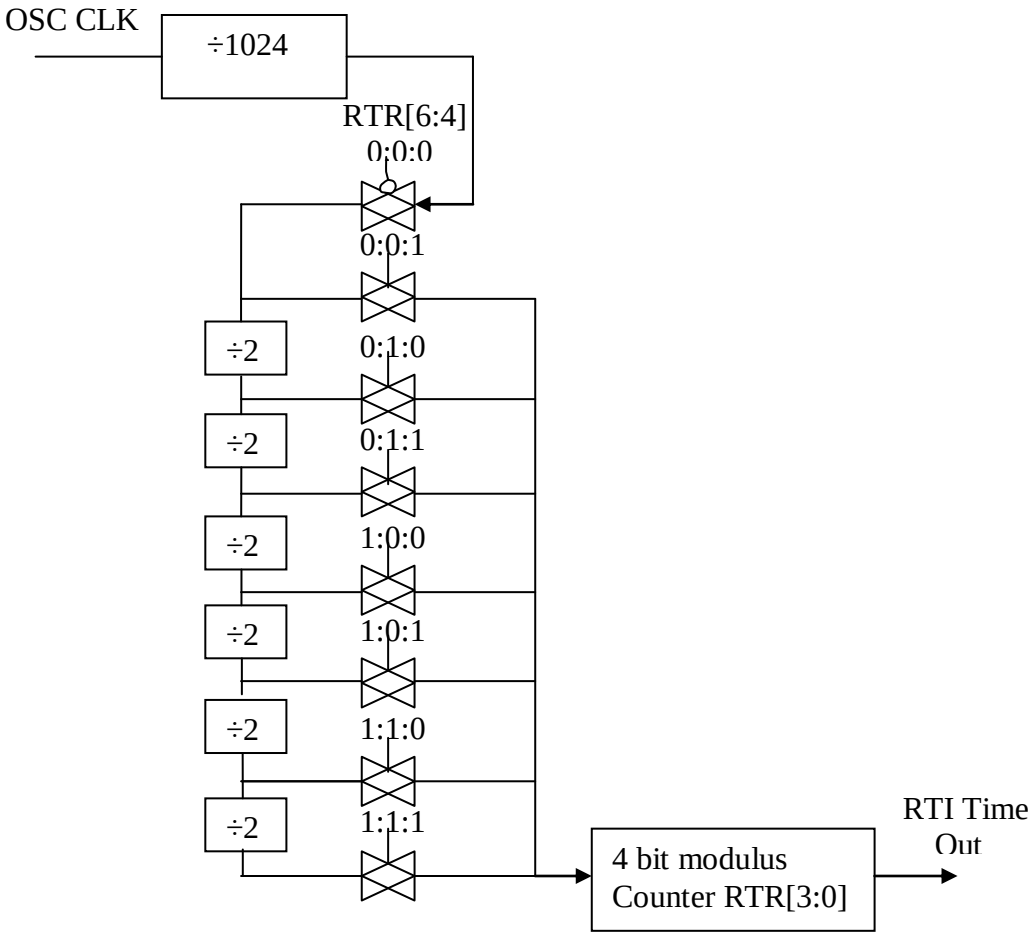


Fig 3.5 Clock chain for RTI

3.7 MOTOROLA SCALABLE CAN:

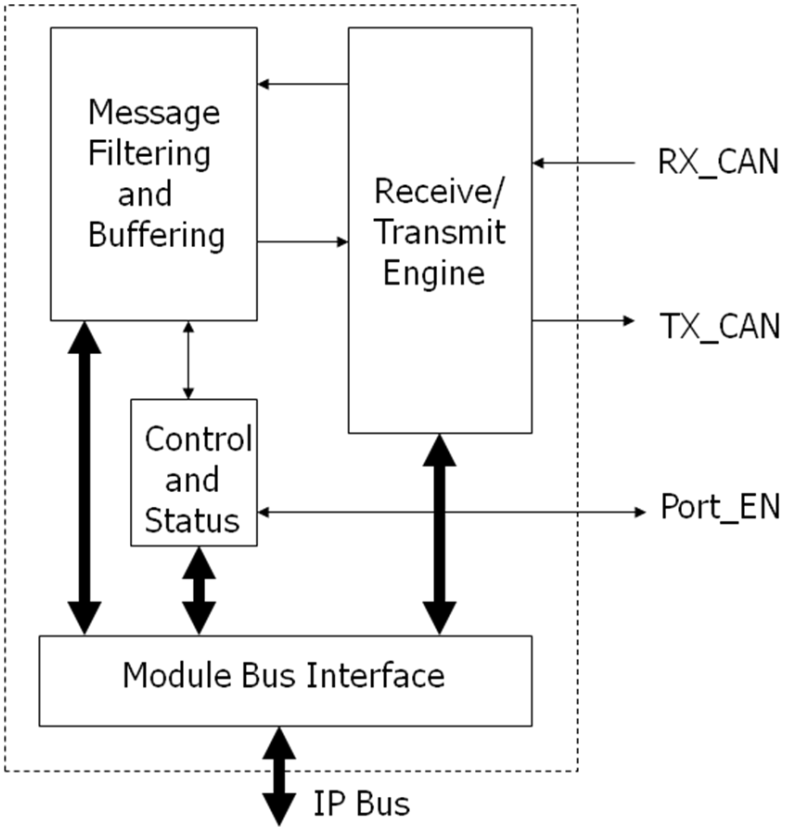


Fig 3.6 MSCAN block diagram.

The MC9S12DP256B microcontroller released by Motorola includes five CAN 2.0 A, B software compatible modules, primarily, but not only, designed to be used as a vehicle serial data bus. The block diagram of a CAN module is shown in figure above.

3.7.1 FEATURES

- The main features of MSCAN includes,
- Can implement version A/B of CAN protocol
- Four FIFO receive buffers
- Tree transmit buffers
- Flexible maskable identifier filter
- Programmable wake-up functionality

- Loop back mode for self test operation
- Programmable listen only mode
- Two different clocks source either system clock or crystal oscillator output
- Three error states for the nodes, Warning, Error passive, and Bus-off
- Global initialization of configuration registers.

3.7.2 TRANSMIT STRUCTURE

There are three transmit buffers arranged in a CAN module, having thirteen byte data structure similar to the outline of the receive buffers. It contains an additional Transmit Buffer Priority Register and two bytes of time stamping register. While transmitting a message the CPU identifies any transmit buffer empty or not. If a transmit buffer is available, CPU stores the identifier, control bits and data contents to the buffer and flagged as ready for transmission.

3.7.3 RECEIVE STRUCTURE

The receive structure contains a four stage FIFO receive buffer, alternatively mapped in to a single memory area. Each buffer has a size of fifteen bytes to store control bits, identifier, data contents, and time stamp bits. Received message will be stored in a receive background buffer which is associated with MSCAN and will passes through the identifier acceptance filter. If the identifier is matching the message will be transfer to foreground receive buffer and sets the receive flag.

3.7.4 IDENTIFIER ACCEPTANCE FILTER

The identifier acceptance filter contains two sets of registers, one is identifier acceptance register the other is identifier mask register, and both have eight bytes of size. The filter can programmable to operate in four different modes:

First mode is, configuring as two 32 bit identifier acceptance filters. If the extended or standard message identifier is matching with first set of filter (CANIDAR0-3, CANIDMR0-3) then a filter 0 hit will be produced and if the message identifier is matching with second set of filter (CANIDAR4-7, CANIDMR4-7) a filter 1 hit will be produced.

Second mode is, configuring as four 16 bit identifier acceptance filters. Here first filter bank (CANIDAR0-3, CANIDMR0-3) produce filter 0 and filter1 hit and second filter bank (CANIDAR4-7, CANIDMR4-7) will produce filter2 and filter3 hit.

The third mode is eight identifier acceptance filters. First filter bank will produce filter0 to filter3 hits and the second filter bank produce filter 4 to filter7 hits.

Fourth mode is closed filter. In this mode no CAN message will be stored in the foreground buffer.

3.8 HCS12 FUZZY ENGINE

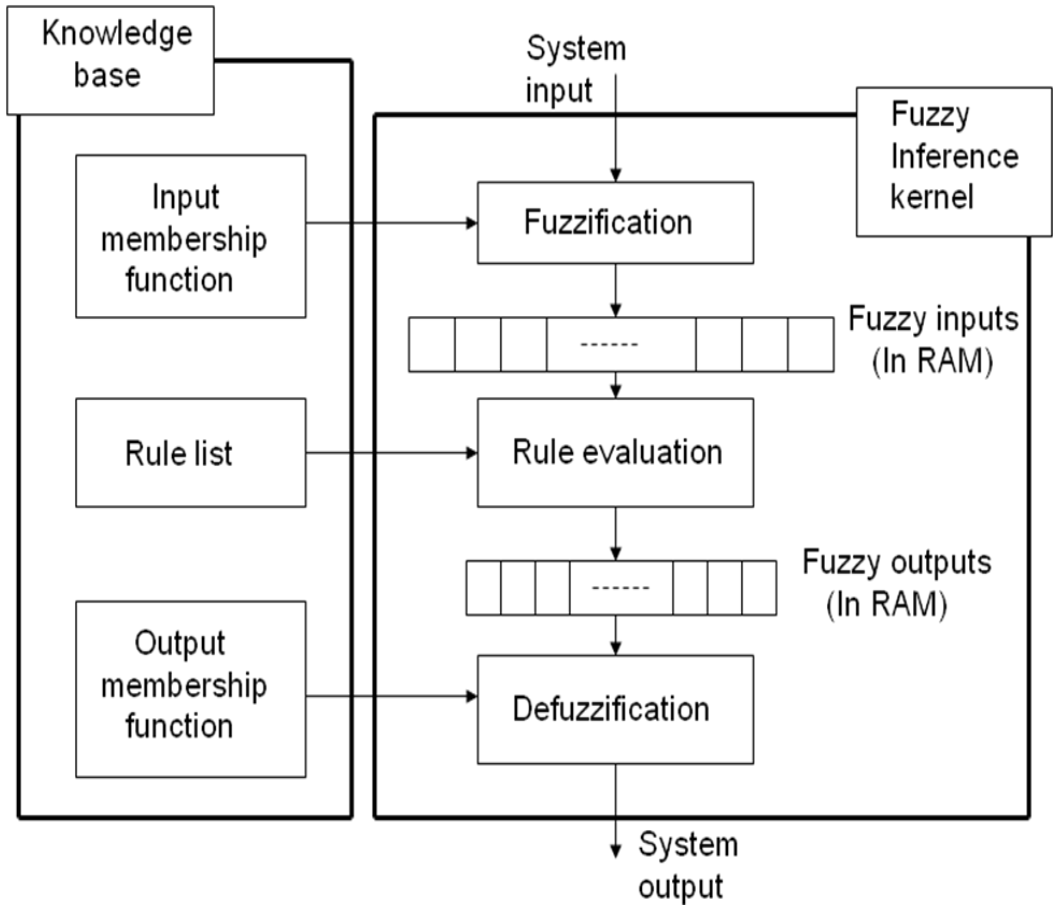


Fig 3.7 Block diagram of MSCAN Fuzzy logic Support

The microcontroller based fuzzy system has two parts: a knowledge base and a fuzzy inference kernel. The knowledge base consists of input membership functions, rule list, and output membership functions stored in non-volatile memory. Membership functions are

the expressions showing the degree of truth determined by the experts. CPU 12 typically uses trapezoidal membership function for input and singleton for output. The membership function for each label requires four bytes of space in non-volatile memory. First byte contains the x-axis starting point of the leading side, second byte contains the slope of the leading side, third byte contains the right most point in the x-axis, and the fourth byte contains the slope of the trailing side. Before starting fuzzy operations we also want to store rule list in the non-volatile memory. The 8 bit offset value of all antecedents of first rule store one by one in memory, then the 8 bit offset value of the consequent part of the rule, which is the fuzzy output of the corresponding rule, will be stored separated by a character '$FE' from the antecedent part. Then remaining rules also will store in the same manner consecutively in the memory. Each rule is separated by the same character '$FE'. End of the rule list is specified by storing another character '$FF'.

The next part of fuzzy system is the fuzzy inference kernel which performs the three major steps, fuzzification, rule evaluation, and defuzzification, for a fuzzy operation. The fuzzy inference kernel uses the following instructions for its operation,

1. MEM (Fuzzification):

The fuzzification step uses instruction MEM, during this instruction CPU compare the system input value with the stored membership function, and find the degree of truth, which is the fuzzy input to the system. For each label in the membership function we have to execute separate MEM instruction. The fuzzy input which gets after each MEM instruction will be stored in consecutive RAM locations. Before doing the instruction MEM the prerequisite is to store the system input in the accumulator A, and initialize the pointers X and Y by storing the address of membership function to X and fuzzy input location to Y.

2. REV/REVW (Rule Evaluation):

Rule evaluation is the central element of fuzzy inference kernel, which gives the fuzzy output by processing fuzzy inputs according to the rules in the rule list. CPU 12 provided two types of rule evaluation instruction, REV and REVW. REV is an unweighted rule evaluation method and REVW is weighted rule evaluation, which allows each rule to have a separate weighting factor stored in the rule list. The REV instruction implements basic max-min evaluation. Here a fuzzy *AND* operator do the min operation and a fuzzy *OR* operator do the max operation. The fuzzy *AND* operator takes the smallest (minimum) antecedent as the truth value of the corresponding rule and if any other rule affect the same fuzzy output, then fuzz *OR* (maximum) operator will take the most true value which will be stored in the fuzzy

output. The index register X contains the address of the first element in the rule list and Y contains the address of the fuzzy input. The accumulator A is used for intermediate calculations by the CPU therefore its need to be set $FF before executing REV instruction. The V bit in condition code register (CCR) indicates whether antecedents or consequents are being processed. So as another prerequisite for the instruction we have to clear V bit and also we want to clear the fuzzy output to $00.

3. WAV (Defuzzification):

The final step in the fuzzy logic program is the defuzzification which computes a crisp output from the fuzzy outputs. The instruction WAV is used to do the defuzzification operation, which process on the singleton output membership function to get a crisp output. The fuzzy outputs correspond to the y-axis value of the output labels.

$$\text{System output} = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

The formula for calculating crisp output is shown above, here n is the number of labels for system output, S_i are the singleton positions, and F_i are the fuzzy outputs from RAM. The WAV instruction calculates the numerator and denominator part in the formula. The division operation is performed by a separate EDIV instruction followed immediately after the WAV instruction. Before executing WAV instruction the accumulator A should be loaded with the number of iteration *n*.

CHAPTER 4

IMPLEMENTATION

4.1 INTRODUCTION

The aim of this thesis work is to implement a three node CAN for automotive applications. The three nodes for this CAN network are, Anti-Lock braking system (ABS), Adaptive Cruise Control (ACC), and seat belt module. To implement this, we have made a vehicle setup, consisting of a vehicle which is rotating on a circular track. ABS is using the input signals from two wheel speed sensors, attached to the front and rear wheels of the vehicle, and a signal from the brake pedal for its operation. Here we are not considering different road condition for the ABS operation because of our limitation in our vehicle set up. ACC module monitor forward going vehicle and provide proper signals to the braking module. We have attached an LED to illustrate the working of seat belt module, instead of an actual seat belt mechanism. We have used three HCS12 family microcontroller boards for processing each node.

4.2 VEHICLE SETUP

We have made our own setup to implement our applications. The vehicle used here is a basic electric vehicle. It uses a dc motor to move the vehicle on its track. The rear wheels of the vehicle are connected to the motor shaft and front wheels are freely rotating according to the vehicle movement. We have attached two optical wheel speed sensors to the vehicle one to the rear wheels and the other to the front wheels. The sensor attached to the rear wheels gives the tire speed, and the sensor attached to the front wheels gives the vehicle speed. These sensor outputs are taking as the input signal by the ABS module for it operation. One infrared distance sensor, using for ACC operation is fixed at the front side of the vehicle. A simple PWM circuit is using to control the Speed of the motor. This circuit includes a 555 timer as an astable multivibrator, and the pulse width of output square wave is varying using a potentiometer. So consequently the potentiometer knob is acting as the brake pedal for our system. A stepper motor is also attached to this potentiometer knob which will give a vibrating rotation to the potentiometer while ABS in operation. A driver circuit using Darlington pair transistor TP122 amplify the current of the signals coming from the microcontroller board to drive the stepper motor.

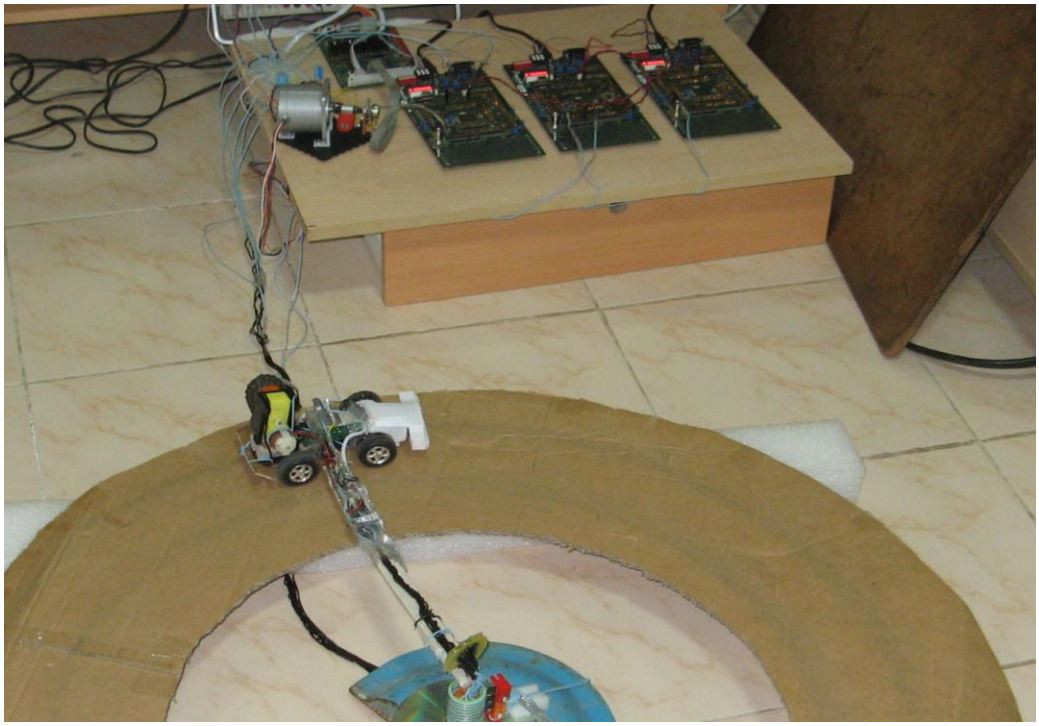


Fig 5.1 vehicle set up

4.3 SENSORS

4.3.1 WHEEL SPEED SENSOR

The sensor module consists of an infrared reflector sensor and a code wheel sticker. The infrared light emitted from the sensor will reflect back after hitting the black and white sticker. A photo transistor in the module will receive little or more reflected light depends on reflection surface. The light intensity comes from the black area will be less and it cannot switch on the transistor that will cause a ‘Logic 1’ output. The reflected light from the white area make the transistor on and the resulting sensor output will be ‘Logic 0’. When the wheel finishes a complete round, a total of nine pulses will get at the output of the sensor.

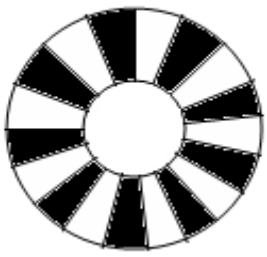


Fig 4.2 code wheel sticker

4.3.2 DISTANCE SENSOR

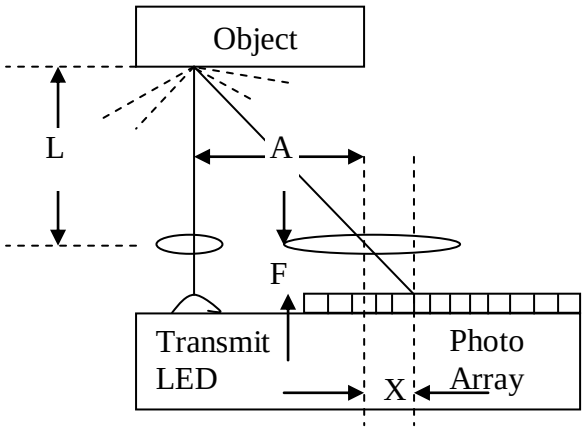


Fig 4.3 Infrared sensor functioning

The infrared distance sensor transmits light to the object in front. The light will pass through a condense lens so that the light intensity is focused to a point. The reflected light will capture through another lens to determine the point of impact. And this can be calculated using the formula:

$$\frac{L}{A} = \frac{F}{X}$$

This sensor can measure a range from 4 to 80cm and the output voltage is ranging from 0.4V to 2.4V when supplied by +5V. An acknowledge period of 32 to 52.9 ms is required before reading the output.

4.4 SPEED CONTROL CIRCUIT FOR DC MOTOR

This speed control circuit uses a 555 timer IC wired as astable multivibrator keeping constant frequency at the output (1.2 KHz) and variable duty cycle. A 22K ohm potentiometer is using to control the duty cycle of the output. When the potentiometer is in its up position, capacitor C2 will charge through resistance R1 and the timer output will be VCC. If the capacitor voltage reaches threshold voltage at the threshold pin of the timer discharge transistor in the timer will go to ON state cause capacitor to discharge through resistor R2 and R3. This will make the timer output at zero level. This will give a pulse output of 5% duty cycle. The potentiometer in its down position cause charging of capacitor

through R1 and R2 and discharging through R3. Consequently the output will have 95% duty cycle.When potentiometer in its 50% position charging and discharging resistance will be equal; this will give a pulse of 50% duty cycle at the output.

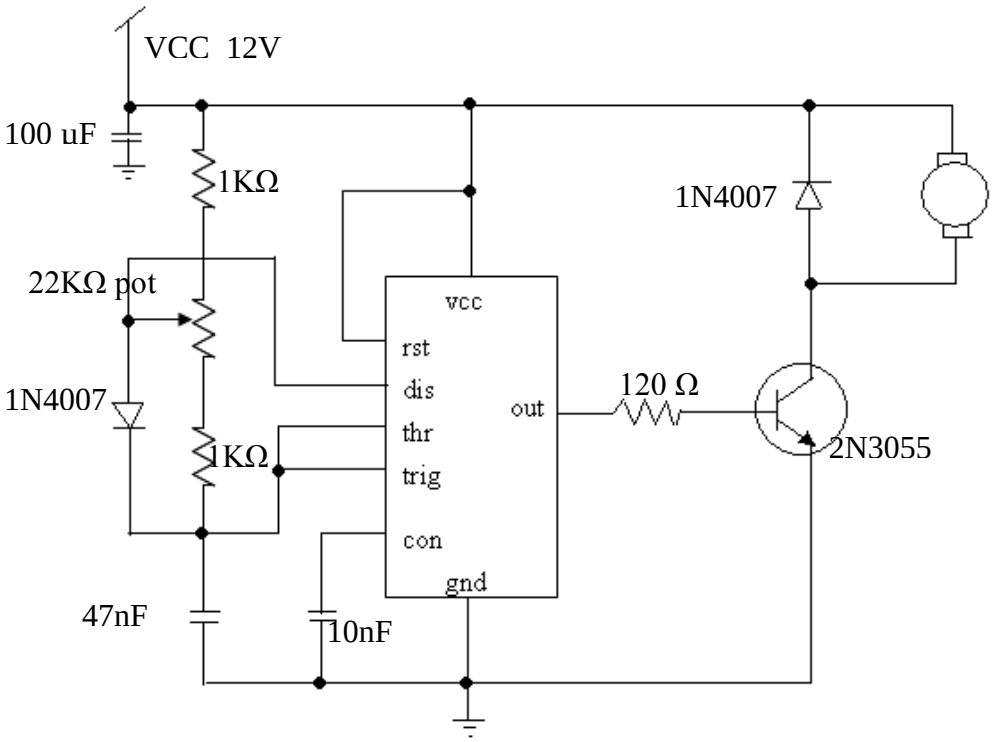


Fig 4.4 DC motor speed control circuit

4.5 CAN BUS

Three MC9S12DP256B, Motorola evaluation boards are acting as the three application nodes for the CAN network. The three nodes are connected by twisted pair cable to a common serial CAN bus with termination resistance of 120 Ω. The bus lines are called CAN_H and CAN_L, and are driven by the nodes with a differential signals. This makes the CAN bus to insensitive to electromagnetic interference.

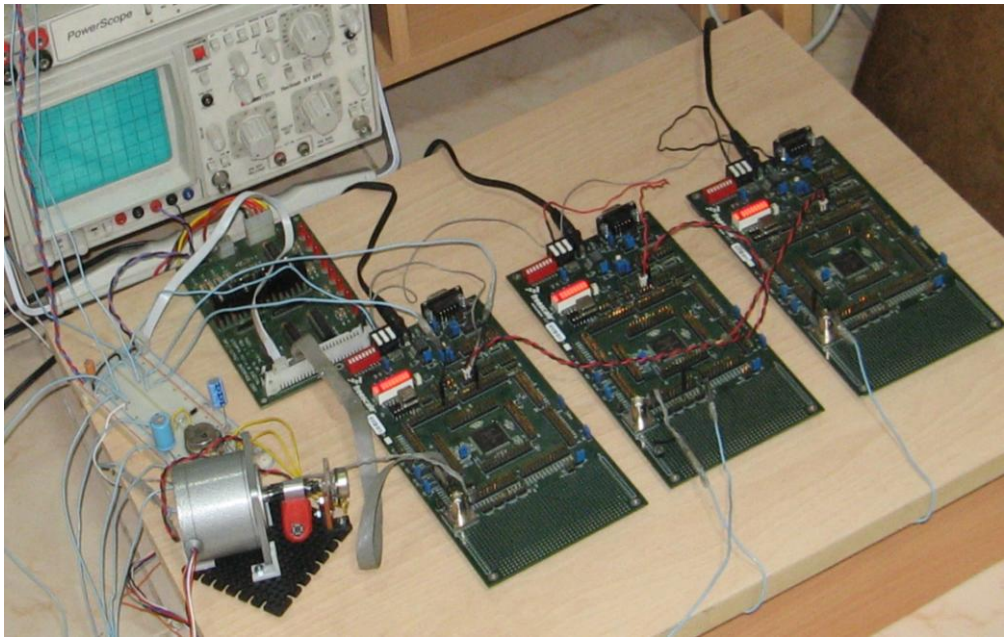


Fig 4.5 CAN network

4.5.1 TRANSCEIVER (MC3388D)

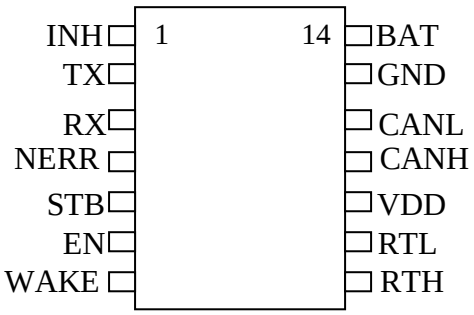


Fig 4.6 Transceiver Pin Connection

The Motorola transceiver MC33388D converts the digital CAN message to differential signals for the communication through the network. The pin diagram of the IC is shown above. In our evaluation board setting of jumpers 15 and 16 connects the CAN module 0 and the Transceiver, which is shown below. Jumper-15 controls the control inputs and outputs for CAN 0 interface. Jumper-16 is the CAN 0 physical interface connector.

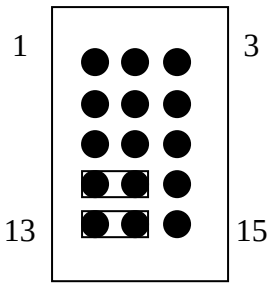


Fig 4.7 Jumper-15

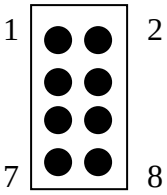


Fig 4.8 Jumper-16

Pin connection of jumper-16 is as follows:

1. GND 2. GND
3. CAN_H 4. CAN_H
5. CAN_L 6. CAN_L
7. GND 8. GND

4.5.2 CAN CONTROL AND STATUS REGISTERS

This section explains the main control and status registers which we have used for the CAN module in our application.

CAN0BTR0 (MSCAN Bus Timing Register 0):

SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0

Bit 7 Bit 0

Synchronization jump width (SJW1-0) = 2 time quanta (Tq)

Baud rate prescaler value (BRP5-0) = 8

Register content = 0100 0111 ($47)

CAN0BTR1 (MSCAN Bus Timing Register 1):

SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10

Bit 7 Bit 0

Sampling (SAMP) = one sample per bit

Time segment 2 (TSEG22-0) = 5 Tq clock cycles

Time segment 1 (TSEG13-0) = 9 Tq clock cycles

Register content = 0100 1001 ($49)

$$\text{Bit Time} = \frac{\text{Prescale value}}{f_{CANCLK}} \bullet \text{No of Tq}$$

$$= 8 * 16 / 16\ \text{MHz} = 8\ \mu S$$

CAN0IDAC (MSCAN identifier Acceptance Control Register):

0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0

Bit 7 Bit 0

Identifier acceptance mode (IDAM1-0) = Four 16 bit Acceptance Filters

Register content = 0001 0000 ($10)

CAN0CTL0 (MSCAN Control 0 Register):

RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ

Bit 7 Bit 0

CAN Stops in Wait mode (CSWAI) = the module ceases to be clocked in wait mode

Wake up Enable (WUPE) = MSCAN is able to restart

Register content = 0010 0100 ($24)

CAN0CTL1 (MSCAN Control 1 Register):

CANE	CLKSRC	LOOPB	LISTEN	0	WUPM	SLPAK	INITAK

Bit 7 Bit 0

MSCAN Enable (CANE) = the MSCAN module is enabled

MSCAN clock source (CLKSRC) = Oscillator clock

Wake up mode (WUPM) = Wakes up only in case of a dominant pulse on the bus which has a length of Twp and WUPE equal to 1 in CAN0CTL0.

Register content = 1000 0100 ($84)

CAN0RIER (MSCAN Receiver Interrupt Enable Register):

WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE

Bit 7 Bit 0

Receiver Full Interrupt Enable (RXFIE) = a receiver buffer full event causes an interrupt

Register content = 0000 0001 ($01)

4.6 ANTI-LOCK BRAKING SYSTEM

The first node, which is acting as ABS module will take the signals from wheel speed sensors and brake pedal. Timer module in the microcontroller counts the pulse width of the signals coming out of the wheel speed sensors. The tire wheel speed sensor output is connected to Timer channel 0, which is configured as input capture. The vehicle speed sensor is connected to Timer channel 7, which is configured as pulse accumulator. We are measuring the braking time here to get brake pressure indirectly, which is using in real vehicles. Braking time is the time the brake pedal (potentiometer knob) takes to reach its down position once applies the brake. If it takes more time we can conclude pressure applied is less, and, if takes less time then the applied braking pressure is high. A real time clock interrupt has enabled in the algorithm to count the braking time duration. The three inputs, which are taking by the ABS module, will go to the fuzzy engine of the microcontroller to do the necessary steps for the operation.

4.6.1 FUZZY BLOCK

Fuzzy operation basically includes three steps, fuzzification, rule evaluation, and defuzzification. The fuzzification operation converts the crisp input to fuzzy values. The microcontroller uses an instruction MEM to do fuzzification. By evaluating speed count at different potentiometer position we have created membership function for each input, which can see from the figure below.

Here input membership function for wheel speed sensors has three labels High, Medium and Low. After fuzzification we will get fuzzy output in RAM locations for each label. The speed variation of motor speed at various parts of sloppy road arises nonlinearity in our wheel speed count. Fuzzy logic is helping us to determine the slip of the vehicle in this nonlinear condition. The rules which determine the slip is given below. We have used instruction REV for rule evaluation, which is an unweighted rule evaluation scheme supported by the microcontroller. The fuzzified brake time and fuzzy slip output will be used to determine the step number required to give to the stepper motor to control the brake. A second set of fuzzy rules are using here to evaluate the fuzzy output.

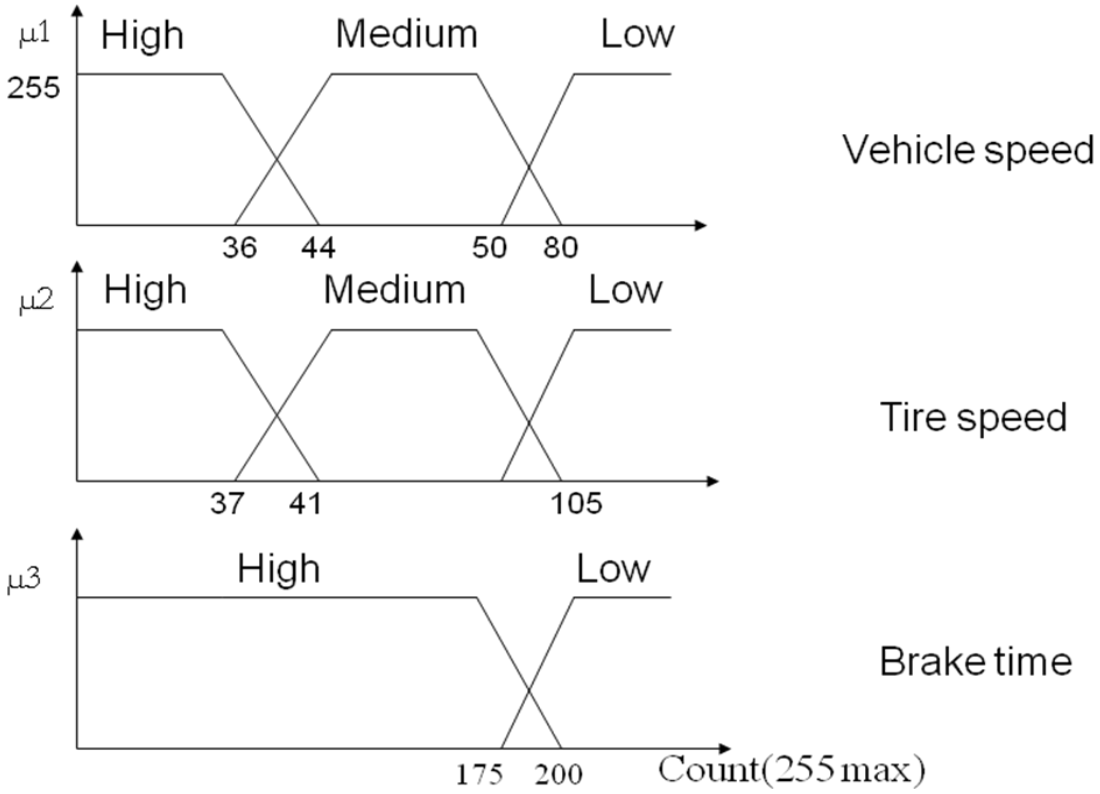


Fig 4.9 input membership functions

4.6.2 FUZZY RULES

Slip Ratio

IF vspeed is med AND tspeed is low THEN moderate slip

IF vspeed is med AND tspeed is med THEN minimum slip

IF vspeed is high AND tspeed is low THEN maximum slip

IF vspeed is high AND tspeed is med THEN moderate slip

IF vspeed is high AND tspeed is high THEN minimum slip

No of Steps

IF slip is min AND brake is low THEN step0

IF slip is min AND brake is high THEN step1

IF slip is mod AND brake is low THEN step1

IF slip is mod AND brake is high THEN step2

IF slip is max AND brake is low THEN step2

IF slip is max AND brake is high THEN step3

Last operation of the fuzzy engine is to generate a crisp output from the fuzzy output stored in its output RAM location. A singleton output membership function, which already stored in the ROM location of the microcontroller, is used by the WAV instruction to find the crisp output. The output membership function which we have used is shown below.

Fig 4.10 Output membership function for ABS

The crisp output (no of steps) coming out of the fuzzy engine uses to rotate the stepper motor in clock wise direction and back. As we want to give an apply release apply mechanism for ABS operation we are applying the step value continuously to stepper motor by dividing the step output with a constant maintaining a proper slip until step value become zero.

4.7 ADAPTIVE CRUISE CONTROL

A simple infrared sensor uses here to detect any forward vehicle or obstacle on the front line of the vehicle. If it detects any obstacle near to its clearance gap it sends a message to the ABS module which will stop the vehicle maintaining the clearance gap. If again it detects no obstacle in that region it will send another message to ABS module to accelerate the vehicle to its previous speed.

4.7.1 FUZZY BLOCK

Fuzzy block in ABS module determines how many steps the stepper motor has to rotate to stop the vehicle. After getting the message from the ACC module the fuzzy block will take the speed of the vehicle and determine the step value. The output membership function and the rules for this fuzzy block are shown below.

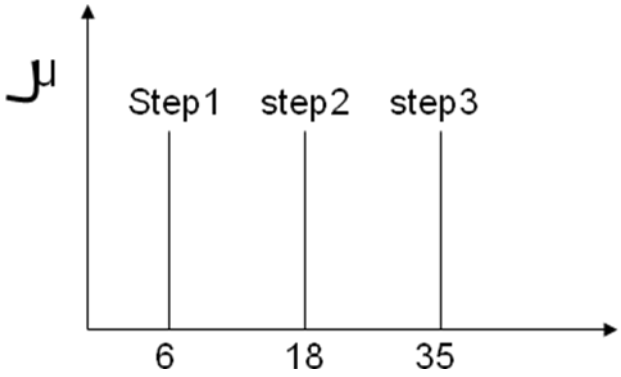


Fig 4.11 Output Membership Function for ACC

4.7.2 FUZZY RULES

IF speed is low THEN step1

IF speed is med THEN step2

IF speed is high THEN step3

The stepper motor using here is two phase permanent magnet type motor. The stepping scheme used here is 2-phase scheme; table below shows the switching sequence in 2-phase scheme.

Clockwise					**Anti clockwise**				
Step	A1	A2	B1	B2	Step	A1	A2	B1	B2
1	1	0	0	1	1	1	0	1	0
2	0	1	0	1	2	0	1	1	0
3	0	1	1	0	3	0	1	0	1
4	1	0	1	0	4	1	0	0	1

Table 4.1 2-phase switching scheme of stepper motor

The stepper motor connected to lower nibble pin of the port H, PTH. Step value 1 means microcontroller will send the full sequence in the table once to the PTH with some delay between two data. So the step value from fuzzy block determines number of time the sequence has to repeat.

4.8 SEAT BELT MODULE

For the seat belt module we don't have any seat belt mechanism in our vehicle, instead we fixed an LED in the vehicle which will glow when it get message to activate the module. It will get its message from the other two modules to when sudden braking occur.

4.9 MESSAGE COMMUNICATED

Our communication network uses four messages for the complete operation. The messages are:

1. Brake applied
2. Critical distance
3. Safe distance
4. Stop mod3.

Table below shows message communication diagram. When we press the brake pedal a message 'brake applied' will transmit to seat belt module to switch on LED. After completing the ABS operation it will send another message 'Stop mod' to node 3 to inform that brake has successfully applied. When ACC module is switched on it will monitor the forward obstacle continuously and if it detects any in its clearance gap it will send a message 'Critical distance' to ABS module. If the vehicle comes back to its safe region another message 'Safe distance' will be transmitted to the ABS module.

Messages:	Node 1 (ABS)	Node 2 (ACC)	Node 3 (SB)
Brake applied	Transmitter	--	Receiver
Critical distance	Receiver	Transmitter	Receiver
Safe distance	Receiver	Transmitter	--
Stop mod3	Transmitter	--	Receiver

Table 4.2 Message communication chart

4.10 RESULTS

This section shows the Data Frames which got in the Logic Analyzer and it has verified according to the CAN protocol. The Logic analyzer input has connected to transmitter and receiver pins of each node. The details of HEX values used as the messages are given below.

1. Brake applied $BB
2. Critical distance $DF
3. Safe distance $D2
4. Stop mod3 $88

The selection of hex value is by own, and can choose any value as data.

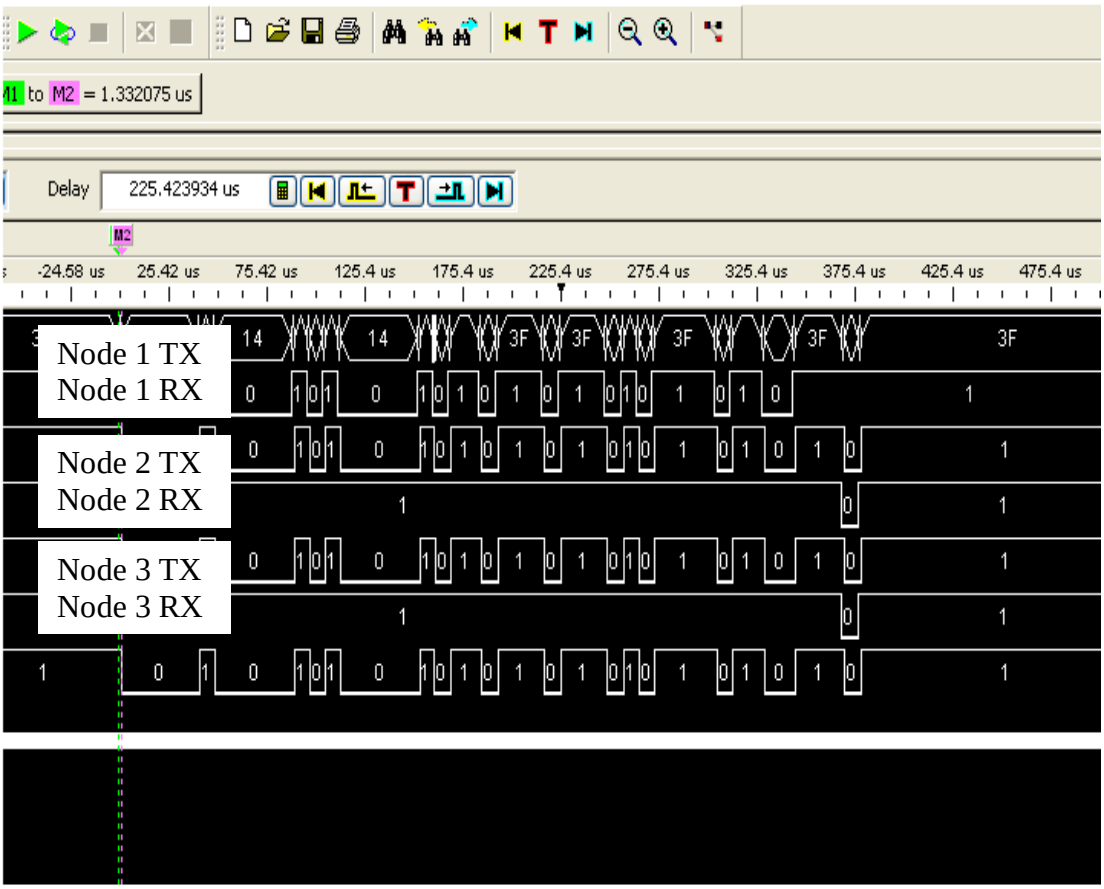


Fig 4.12 message $BB Data frame

The logic analyzer window in this figure 4.12 shows the data frame used by the network for the message $BB. This shows the pin status of the transmitter and receiver pin of all the nodes. Here the message is transmitted by the Node 1 and all the receiver pins gets the message. The message passes through the identifier filter section of all the receivers then, and the message accept by the receiver with matching identifier. The marker in this window shows the delay or latency between the transmission and the reception by any of other node, here it is about 1.3µs.

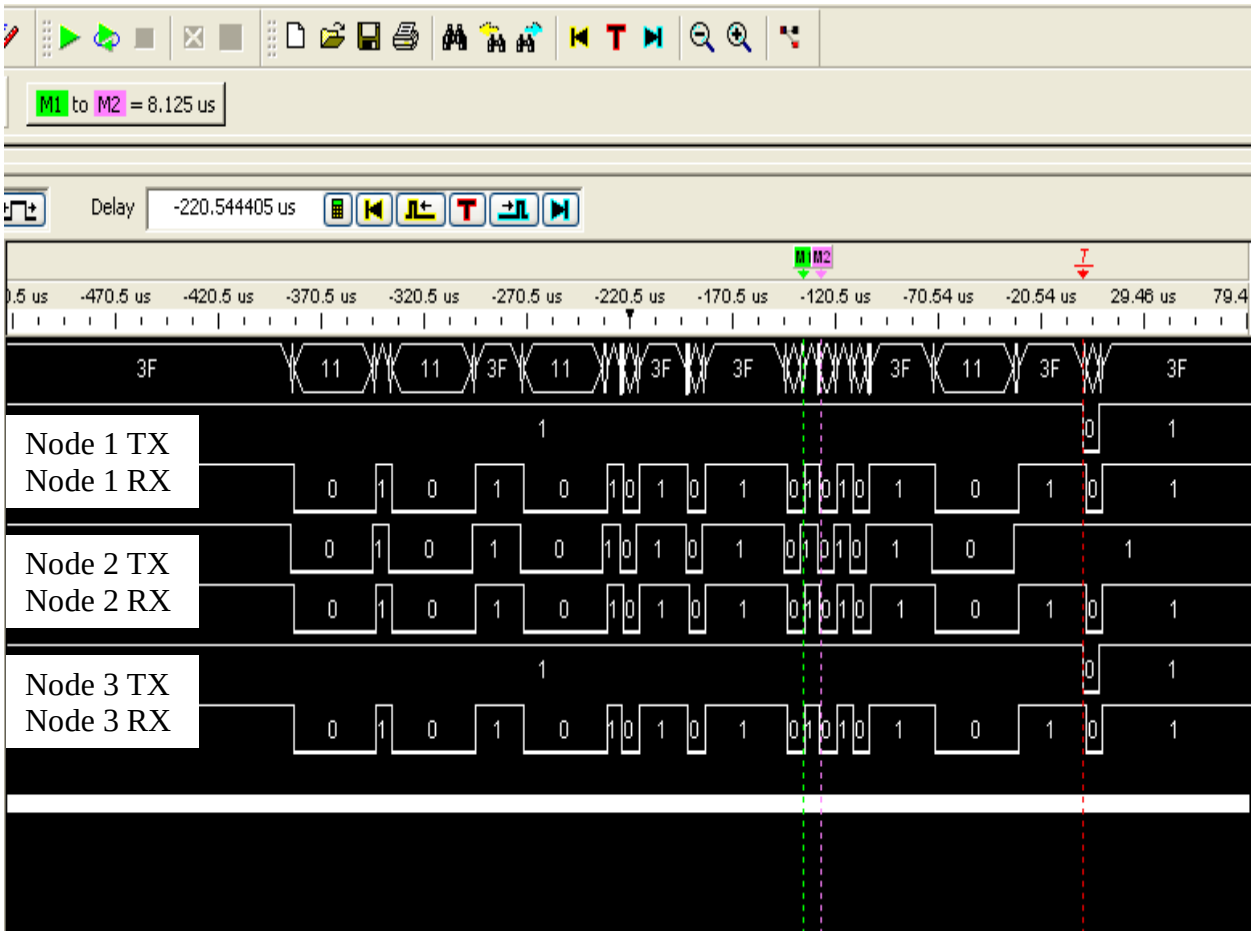


Fig 4.13 message $DF Data frame

Here the message transmitted is $DF. The transmitter here is the Node 2, and it has sent message while the vehicle enters the clearance gap area. After the successful reception of the message the receiver has sent an acknowledgement signal through its transmitter line. This can see from the window above. The markers here shows one bit period of the message frame. The value can see in the top left corner of the window and is equal to 8.125µs.

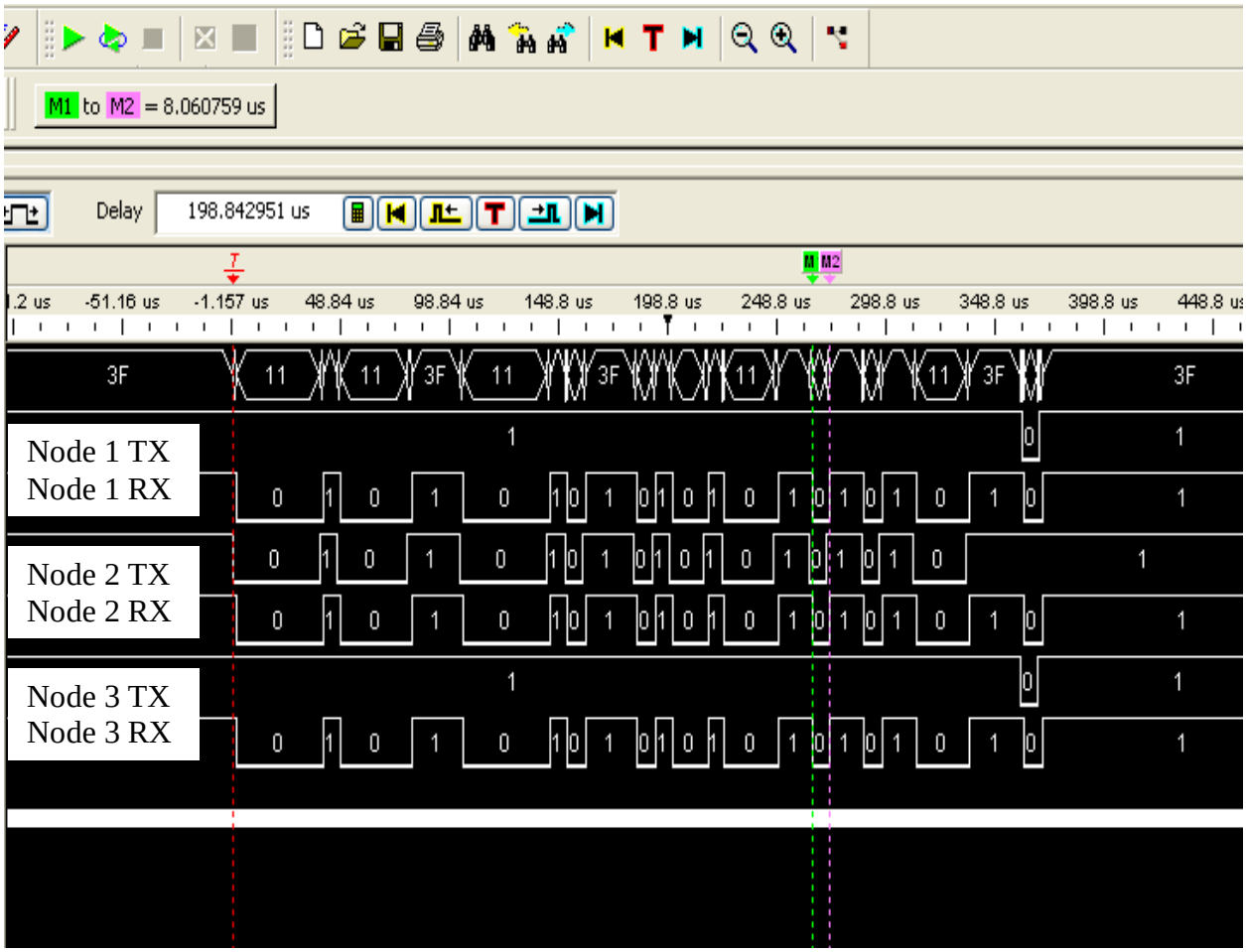


Fig 4.14 message $D2 Data frame

The message $D2 sent by the Node 2 can see from figure above. Here the receiver is Node 1. After the Node 1 received the correct message it has transmitted an acknowledgement dominant bit in the acknowledgement slot. Here again the marker shows the bit period, which is equal to 8.06µs. The bit time period set by the program is 8 µs.

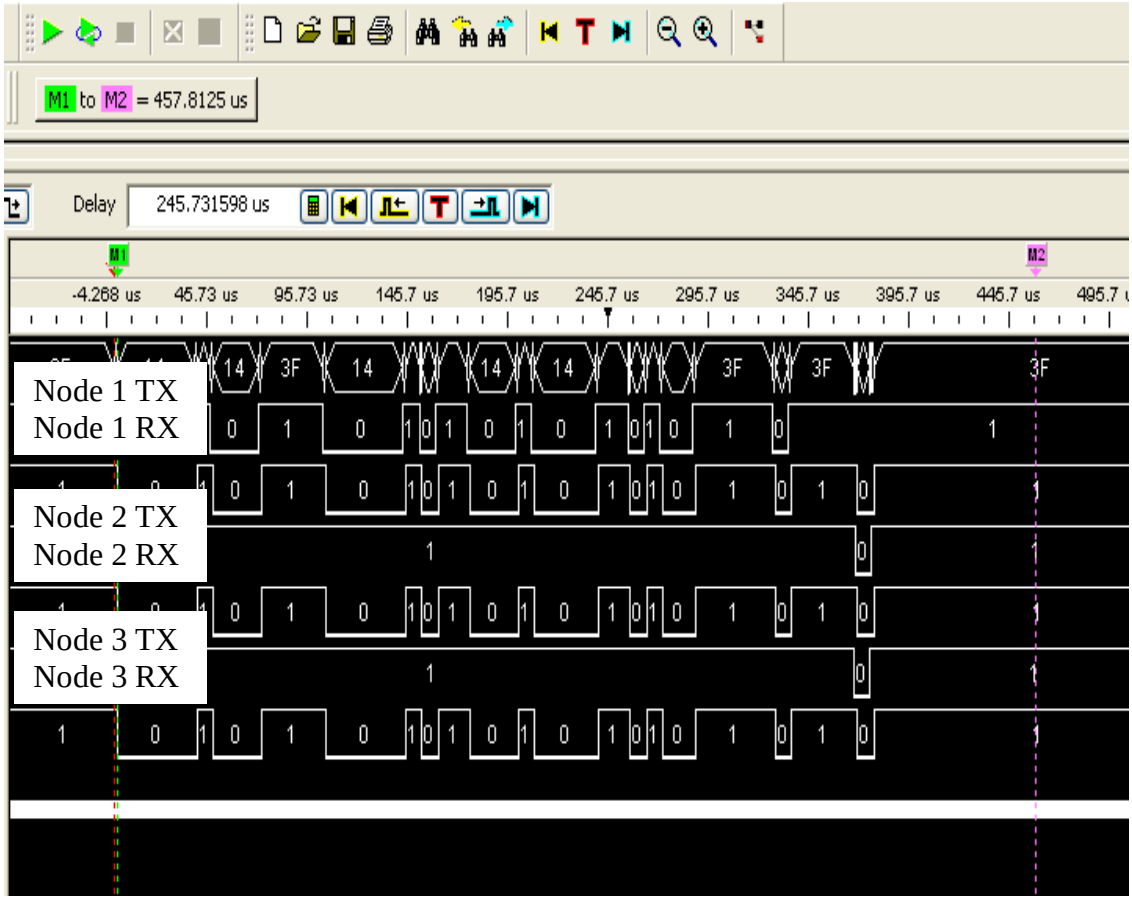


Fig 4.15 message $88 Data frame

The logic analyzer output window for the message $88 is shown above. The marker in this figure has kept to measure the total time duration taken to transmit a complete message. The message here has one byte data field and it also includes some stuff bits.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This thesis work concentrates on implementing a reliable three node CAN network for automotive applications. Anti-Lock braking system, Adaptive Cruise Control, and Seat Belt module on a simple dc motor controlled electric vehicle forms the three automotive nodes for the network. The network has implemented using three Motorola MC9S12DP256B microcontrollers. Here first chapter is an introduction to the problem which has been addressed in this work. This discusses some overview about Controller Area Network, Anti-Lock braking system, Adaptive cruise control, and some fuzzy concepts. The entire second chapter is dedicated to CAN protocol. It deals with the CAN specifications and standards, its properties, its efficiencies, its advantages and disadvantages. Chapter 3 gives a brief idea about the Motorola HCS12 microcontrollers; STAR12 CPU, features, on-chip peripherals, fuzzy block, and Motorola Scalable CAN (MSCAN). Finally implementation details of this thesis work explain in Chapter 4. This portion gives a detailed insight about the vehicle system, how ABS and ACC algorithm implemented using Fuzzy logic support of the microcontroller, how the network is connected, and some communicated message results.

5.2 FUTURE WORK

The problem addressed here is only a first step of CAN protocol to a complex network. This shows how reliably a CAN network functions in a real time system. Considering automotive systems several subsystems relay on networking; such as chassis system, air-bag system, powertrain, body and comfort electronics, X-by-wire, multimedia and infotainment, etc. In future work the CAN network can be applied to all these systems by increasing the number of nodes. For further decentralization for control systems sub network of CAN called LIN (Local Interconnect Network) can be implemented. According to the application different CAN networks have different CAN messaging needs. As for example in powertrain system messages are regular and predictable. So this CAN module requires more number of buffers and increased baud rate. We can implement these types of application using CAN IP's based on FPGA, signal processors other high-end (32-bit) microcontrollers. FlexRay CAN, Time Triggered CAN (TTCAN), CANopen, TouCAN are all different types of network protocol available to achieve the advanced requirements. Apart from automotive applications this protocol can be used in industrial applications, medical equipment field, home automation etc.

REFERENCES

[1]. H. F. Othman, Y. R. Aji, F. T. Fakhreddin, A. R. Al-Ali,"Controller Area Networks: Evolution and Applications". 2006 IEEE.

[2]. Robert Bosch GmbH, "CAN Specification", Version 2.0, September 1991.

[3]. Jadsonlee da Silva Sa, Jaidilson Jo da Silva, Miguel Goncalves Wanzeller and Jose Sergio da Rocha Neto, "Monitoring of Temperature Using Smart Sensors Based on CAN Architecture." Proceedings of the 15th International Conference on Electronics, Communications and Computers, 2005 IEEE.

[4]. Thomas Nolte, Hans Hansson, Lucia Lo Bello, "Automotive Communications – Past, Current and Future". Volume 1, 2005 IEEE.

[5]. Freescale Semiconductor, "MC9S12DP256B Device User Guide V02.15". Freescale Semiconductor, Inc. Jan 11, 2005.

[6]. Freescale Semiconductor, "MSCAN Block Guide V02.15". Freescale Semiconductor, Inc. 15 JUL 2004.

[7]. David E. Nelson, Rajab Chlloo, Robert A.McLauchlan, S. Iqbal Omar, "Implementation of Fuzzy Logic for an Anti-lock Braking system". 1997 IEEE.

[8]. Leon Reznik, Paul Vilas-Boas," Embedded Fuzzy Control for Reefer Refrigeration System". 2001 IEEE.

[9]. Motorola, "Advance information Fault Tolerant CAN interface". Rev: 2.2 Date: 02 Nov 2000.

[10]. Chih-keng CHEN and Ming-Chag SHIH ,"PID Type Fuzzy Control for Anti-Lock Brake Systems with Parameter adaptation". JSME International Journal, series C, Vol. 47, No.2, 2004.

[11]. Todd D. Morton. "Embedded Microcontrollers". Delhi: Pearson Education, 2001.

[12]. Jonathan W. Valvano. "Embedded Microcomputer Systems". Bangalore: Thomson, 2002.