ANALYSIS AND DESIGN OF CONTROLLER AREA NETWORKS

BY

ZHENGOU WANG

Bachelor of Engineering

Chongqing University

Chongqing, P.R. China

1982

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1991

Dhaka
1991

ANALYSIS AND DESIGN OF CONTROLLER

AREA NETWORKS

Thesis Approved :

Huizhu Lu
Thesis Adviser

J Chandler

D. E. Hankins

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my major advisor Dr. H. Lu for her warm encouragement and helpful advisement throughout my graduate study and writing of this thesis.

I am specially grateful to Dr. M. Stone for serving on my committee. He patiently guided me throughout my project, always helping me and giving me exemplary advice when I needed. Without his knowledge and guidance I couldn't have completed my thesis successfully.

My special thanks go to Dr. G. E. Hedrick and Dr. J P. Chandler for their constant encouragement and help. Their suggestions and support were very helpful throughout my masters program.

I would be failing in my duty if I didn't express heartfelt thank to Mr. G. Couger and Dr. M. Kamath for their generous help without which I wouldn't have progressed so well.

My deepest appreciation is extended to my wife, Xinyuan Zhu and my daughter, Zhifeng Wang for their patience and love.

TABLE OF CONTENTS

LIST OF TABLE

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With the continuing decline in the cost of computing, the number of computers and microprocessors used for various process controls, device controls and control systems has increased dramatically. These computers and microprocessors usually do not work in isolation, and with their proliferation comes the demand for suitable communication networks--mainly local area networks, that can interconnect locally distributed computers and/or microprocessors to form rather complex control systems.

Examples of such a demand are: networks in advanced robots which can provide fast communication among arm, vision sensor, proximity sensor control units and other control units so that different parts of the robots can cooperate accurately to carry out more complex tasks; networks in cars which can connect different electrical control units together to reduce the manufacture and maintenance complexities and improve total performances of the cars; networks in tactical airplanes, construction, military vehicles, harvest machines such as combines and chemical reactors which will provide fast communication between various control units and allow various parts of the system to work in a cooperative and optimal manner.

To date, many local area networks have been developed or proposed, but very few can satisfy the above demand. Specifically, the most popular standard local area networks are inadequate for real-time control systems. Their medium access control protocols have certain defects when used for real-time purpose. CSMA/CD (Carrier Sense Multiple Access with Collision Detection) [IEEE85a] type networks are non-deterministic [Hamm86] . Token bus [IEEE85b] and token ring [IEEE85c] type networks are too complex, too expensive and inefficient [Tane89, Stal90].

A controller Area Network (CAN) is a small local area network which is suitable especially for use in real-time distributed control systems. In real-time control systems communication networks play a critical role. Data transmission must be in time. Late delivery of data is a fault that can result in instability or catastrophe.

Data transmission in the network is influenced significantly by the intensity and distribution of traffic in the network. It is subject to time-varying delays due to the latency of messages in the network. In addition, messages may be corrupted by noise in the network medium or lost due to buffer saturation in the receiving stations and, therefore, have to be retransmitted. These problems will increase the message transmission delay further and aggravate the performance of the real-time control systems. For a network such as CAN, which may be shared by processors, each with a different nature, an appropriate traffic load distribution is critical.

In addition to delay considerations, other performance

characteristics such as throughput and channel utilization must be taken into consideration in making any design decision. In fact, performance analysis is very essential for any kind of network design.

The main purpose of this thesis is to analyze the CAN protocol and find out the basic performance characteristics of the CAN. These performance characteristics are the relationships among the message delays, network loads, message priorities and channel utilizations. A simulation program was developed to simulate the network and to verify the analytical results. This simulation program also can be used as a tool to help the network designer to design actual network. Furthermore, the thesis presents a message priority assignment algorithm to help the network designer to determine the optimum message priority assignment. A proof of the correctness of the algorithm is also included in the thesis.

This thesis contains six chapters. The second chapter briefly reviews the most popular local area networks and gives a simple introduction to the CAN. The third chapter provides a more detail description of the communication controller and protocol of the CAN and gives a formal analysis for the CAN protocol. Chapter four presents the message priority assignment algorithm. Chapter five discusses the simulation program and the simulation results. Summary of the thesis and possible future work are presented in the last chapter of this thesis.

CHAPTER II

AN OVERVIEW OF PAST WORK

Local area networks provide communication capabilities among terminals, computers, and other devices within a limited geographical area such as an establishment, a single building, a machine, or a robot. The principal elements that determine the nature of a local area network are: its topology, the transmission medium, and the communication channel access control protocol. These elements are related to each other. They determine the type of data that may be transmitted, the speed and efficiency of communication as well as what kind of applications that a local area network may support [Stal90].

A local area network may have any kind of topology such as ring, star, bus, tree, or an arbitrarily connected graph. A constrained topology such as ring or bus or tree eliminates the need for complex routing switches and store-and-forward buffers, and requires only simple host interfaces. Bus, tree, and ring topologies are therefore generally preferred for local area networks.

A local computer network may connect tens or hundreds of computers, terminals, and other devices. A large number of unrestricted accesses to a shared channel may result in an overlap in time during message transmissions over a single

channel. The occurrence of these garbled message is called a collision. To eliminate collisions or reduce their frequency of occurrence, an access control protocol must regulate the availability of the communication channel to the station interface.

The choice of an appropriate access control mechanism is important in any local area network from the viewpoint of the communication channel's throughput and delay characteristics. There is a broad spectrum of access control protocols which have been proposed for local networks. Among all these protocols, the CSMA/CD, token bus, and token ring which are specified by the IEEE 802 standard, are used widely in various applications.

CSMA/CD

The topology of the CSMA/CD type network is bus or tree. Stations are attached to the bus or tree through interface devices. A station wishing to transmit listens to the bus to determine whether another transmission is in progress. If the bus is idle, the station transmits its message. Otherwise, the station continues to listen to the medium until it is idle, then transmits and checks for a collision. If two or more stations begin their transmission at the same time a collision will occur. After a collision is detected during the transmission, stations will cease their transmission immediately and wait for either 0 or 1 time slot and try again. Here a time slot is a short fixed length of time defined by the communication protocol. If two stations wait the same number of time slots, there will be second

collision. After the second collision each station picks 0, 1, 2, or 3 at random and waits that number of time slots and try to transmits again. In general, after the i-th collision, a random number between 0 and 2^i-1 is chosen and the station waits that number of time slots then try to transmit again. However, after 10 collisions have been reached the waiting interval is fixed at 1023 time slots. After 16 collisions the station will give up [IEEE85a].

According to Hammord and O'Reilly [Hamm86], the main advantage of CSMA/CD is that the entire bandwidth of the channel can be used by a station once it successfully gains access. Under lighter load condition, a station can, on the average, successfully access the channel after a short waiting period.

The disadvantage of CSMA/CD is that, under a given set of conditions, the delay of a message and throughput varies with time. That is, the system is non-deterministic. With a little "bad luck", a message may wait for a very long time before it is transmitted. Further, for a given number of stations, the delay increases and throughput decreases at higher loads because the frequent collisions waste a large portion of the bandwidth.

CMSA/CD is not suitable for use in real-time systems. A real-time system must be deterministic. That is, for a given condition the worst case delay and throughput must be known. Another problem associated with CSMA/CD is that there is no message priority. Important messages may be held up waiting for unimportant messages.

Token Bus

The topology of token bus networks is also a bus or tree. Logically, the stations are organized into a ring, with each station knowing the address of stations preceding and following it. A special control frame, a token, circulates around the logical ring. Only the station holding the token is permitted to transmit frames. Since only one station holds the token at a time, collisions do not occur.

The token bus also defines four priorities classed 0, 2, 4, and 6 for traffic, with 0 the lowest and 6 the highest. When the token comes into a station, the station transfers class 6 messages for certain amount of time. After transmitting the class 6 messages, if the time for the token to stay in that station does not expire the station can transmit class 4 messages. This process is similarly repeated until either all the class 0 messages are transmitted or the time for the token to stay in that station has expired in which case the token will be passed to the next station [IEEE85b].

Unlike the CSMA/CD protocol, the token bus protocol is very complex and requires considerable overhead. The logical ring must be initialized when the network is started. The ring should be able to add or delete stations dynamically. It also must deal with token loss and multiple tokens problems, and it should be able to recover from various failures. The priority scheme guarantees the class 6 messages of every station a known fraction of the network bandwidth, but does not distinguish message importance among stations. All messages in the priority 6 queue in every station are equally important and must wait for the token to

reach the station before getting transmitted [Tane88]

Token Ring

The communication channel of the token ring is a physical ring formed by a collection of individual point to point links. Access to the transmission channel is controlled by passing a permission token around the ring. When the system is initialized, a designated station generates a free token which travels around the ring. When a station needs to transmit frames, it seizes the token, removes it from the ring, and then begins to transmit. At the end of the transmission, the station passes the access permission to the next station by generating a new free token. Because there is only one token, only one station can transmit at a given instant. Thus the token ring solves the channel access problem the same way the token bus solves it.

The token ring has an elaborate scheme for handling multiple priority frames. The token contains a field which holds the priority of the token. When a station needs to transmit a priority n frame, it must wait until it can capture a token whose priority is less than or equal to n. Furthermore when a data frame goes by, a station can try to reserve the next token by writing the priority of the frame the station wants to send into the frame's reservation bits. However, if a higher priority already has been reserved there, the station may not be able to make a reservation. When the current frame is finished, the next token is generated at the priority that has been reserved [IEEE85c].

Token ring networks are simpler than token bus networks, but are still complex and expensive, and have all the overhead problems token bus networks have. In practice a token ring is a star shaped ring with cables coming and going to every station from a wiring center. If the purpose of the network is to reduce the wiring complexity token ring clearly is not a solution [Tane88, Stal90].

Controller Area Network(CAN)

Recently, the number of electronically controlled systems introduced into automobiles has increased significantly . Originally these systems were operated mostly on a stand alone basis. Control units in an automobile measure and process sensor signals redundantly. The wiring, control units, sensors and other components require more and more space. As a result, the complexity of vehicle electronic systems has increased dramatically for manufacturing and maintenance. These problems lead to the development of the so called "in-vehicle networking" [Jurg86].

In 1985, Robert Bosch GmbH (a German company) and Intel Corporation agreed to develop an in-vehicle network device and named the network and the protocol "Controller Area Network" or "CAN" for short. By mid-1987 the first sample product (Intel 82526) was produced [Phai88]. Motorola Corporation has also done some research on this subject. Both Motorola and Intel are designing single chip microcontrollers with a subset of full CAN implementation on the same chip called BasicCAN. Some of these products are available.

CAN is similar to the CSMA/CD type networks except it is much smaller. The communication channel of CAN is a bus. Stations are connected to the bus through communication controller. A station wishing to transmitting message listens to the bus first. If there is no message on the bus then the station transmits its message immediately. If there is a message on the bus, the station does not transmit its message, and continuously listens to the bus. Once the current message on the bus is finished and the bus becomes idle, the station transmits its message immediately. Every message in the system has a pre-assigned message priority. If two or more messages start transmission at the same time the lower priority message loses arbitration and stops. The message with higher priority can be transmitted continuously until it is completed. After the higher priority message is finished, the lower priority message can be transmitted. The above process proceeds repeatedly [Jord88].

CAN has many outstanding features. It is inexpensive, simple, easy to implement, and has a high message rate, short message latency, and no message trashing effect as in standard CSMA/CD based protocols etc. . It is especially suitable for use in robots, planes, cars, construction vehicles, military vehicles, and some industrial real-time control systems. It fills the gap between very low performance UART-type serial links and expensive communication products and protocols such as Ethernet or MAP. According to Intel Corporation's report, the International Standard Organization (ISO) has adopted the CAN as the standard for high speed communication, and it has a good

chance of being the standard for the Society of Automotive Engineers (SAE) [Gupt88]. This means the CAN may have the potential of been world widely used in various vehicles and machines·

Research activities related to the CAN have been reported in recent literature [Inte88, Jord88, Iver88]. Kiencke [Kien86] and Arnett [Arne87] have introduced the CAN to the "Society of Automotive Engineers(SAE)". Phail [Phai88] has also introduced CAN to the "American Society of Agricultural Engineers(ASAE)". Intel Corporation provides a PC based demo board to assist the automotive engineers in understanding and implementing the CAN. This board can be connected to a PC through a RS-232 port that allows programs to be downloaded to or uploaded from the board's RAM. Through the PC, contents of the RAM may be displayed or altered. A microprocessor is included in the board to simulate the host microcontroller [Phai88]. Three sample programs are provided. The user can obtain experience in using the CAN by running the example programs [Manu88].

In 1989 Bickerton and Chauhau [Bick89] in Lucas Automotive Ltd. developed a data bus simulator for use by the automotive designer. This simulator is fully integrated into a LISP software environment. It can simulate CAN protocol. It has an automotive component library and a graphic interface. It supports automotive system level design and assists in the assignment of message priority by simulation and allows a proposed automotive design to be rapidly assessed.

To date, the research activities related to CAN are concentrated on its application in automobiles, mainly in cars. No other applications or formal analysis have been reported in the literature.

CHAPTER III

ANALYSIS OF THE CONTROLLER AREA NETWORK

In this chapter the most important aspects of the CAN communication controller and the CAN protocol are described more detail. The basic performance characteristics of the CAN are studied. In the following discussion the time unit used to express transmission delay is the time needed to transmit one bit which is the inverse of the bandwidth. When discuss message transmission, both the term "message" and "frame" are used, because when transmitted messages must be organized into frames. A message generation can also be viewed as a message arrival to the communication system, so in the following discussion the term "message generation" and "message arrival" are equal.

Structure of Communication Controller

CAN is a single bus, multi-master architecture network. Stations are connected to the bus through a communication controller. Figure 1 shows the structure of the network and figure 2 shows the structure of the communication controller. The most important components of the controller are the on-chip Dual Port RAM(DPRAM), the Interface Management Processor(IMP) and the Processor Interface Logic(PIU). The DPRAM includes global status and a control register and

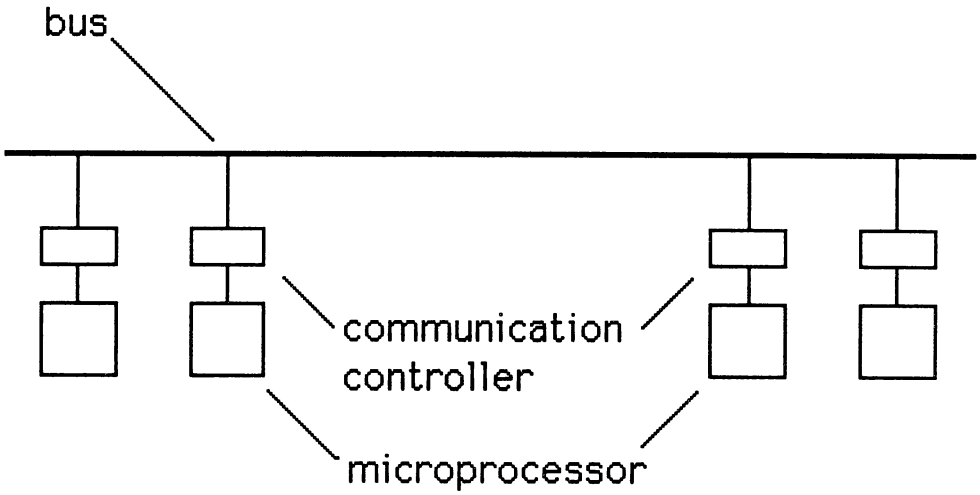
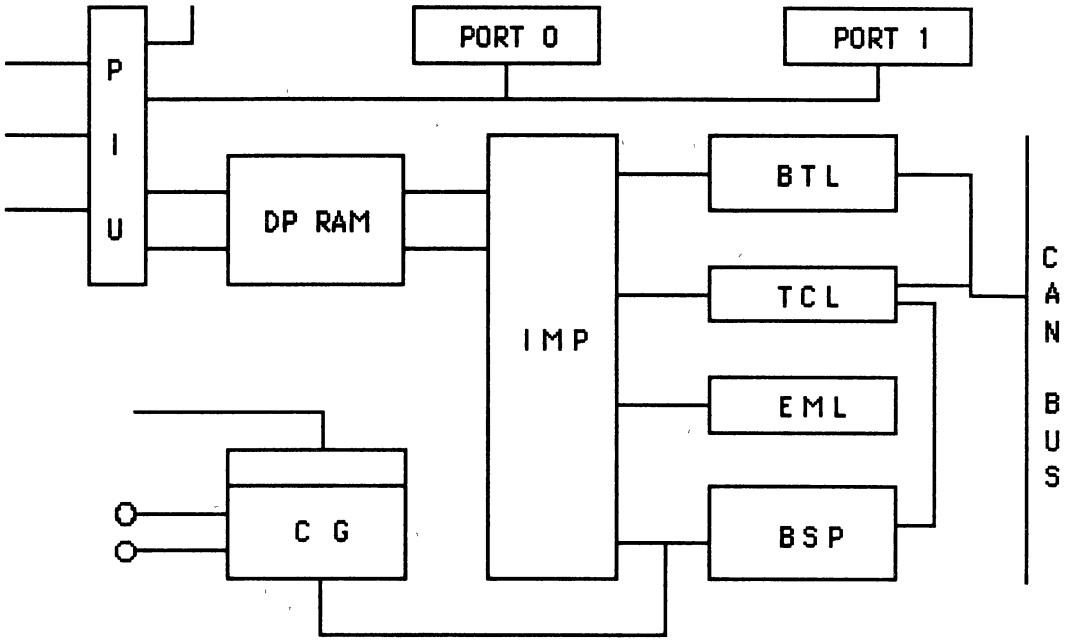


Figure 1. Structure of CAN



PIU : Processor Interface Unit.
CG : Clock Generator.
IMP : Interface Management Processor
BTL : Bus Time Logic.
TCL : Transceive Logic.
EML : Error Management Logic.
BSP : Bit Stream Processor.
DP RAM : Dual Port RAM.

Figure 2. CAN Network Controller Structure

serves as the communication buffer between the station CPU and the IMP. The station CPU treats the DPRAM like its own memory. When the station is powered up, the host CPU initializes the global status and control registers and creates data structures called communication objects in the DPRAM for reception and transmission of defined messages. Later the station CPU will transmit and receive messages by writing and reading the communication objects in the DPRAM. The IMP controls the transmission and reception of data between the serial bus and proper communication object in the DPRAM. The PIU links the DPRAM to the station CPU and it can interface directly with all Intel microcontrollers and microprocessors.

One special future about CAN is that there is no data encoding. Data bits are sent to the bus directly. A binary 1 is represented by a high voltage level and a binary 0 is represented by a 0 voltage level. Unlike the most popular differential encoding scheme there is no clock imbedded in the transmitted data. Each station uses its own clock to receive incoming signals. To obtain correct bit timing, the Bus Timing Logic(BTL) compares the transitions (voltage level changes) in the received signals with the station clock and synchronizes its receiving clock constantly during the reception of frames. The bit stuffing method used in the transmission logic can guarantee enough transitions in the transmitted bit streams. These techniques make the collision detection mechanism of this network very simple.

The CAN Protocol

The data frame structure is shown in figure 3. Each frame starts with a start of frame bit, signaling the start of a data frame. The arbitration field follows the start bit and contains the message identifier and one additional control bit for other purposes. The CAN protocol is a contention based protocol with a prioritized message scheme. Each type of message has a unique message identifier ranging from 0 to 2032 which serves as the name of the message as well as the priority code of the message with 0 the highest priority and 2032 the lowest. When a node wants to transmit a message it checks the bus to see if there is a message on the bus. Here node means the communication controller. If there is no message on the bus the node will begin its transmission immediately. If there is a message on the bus, then the node will wait until the current transmission completed. Once the node finds the bus is idle it will begin its transmission immediately. Transmitted messages are broadcasted to the bus and all nodes on the network are constantly listening to the bus. When a message appears on the bus every node checks the message identifier. If this message identifier matches a receiving object header in a node, then the node will copy this message from the bus to the proper communication object in the DPRAM and sends an interrupt to the station. Later the station CPU will take this message. If this message identifier does not match a receiving object header in this node then the node just simply ignores the message. When two or more nodes begin their transmission at the same time a

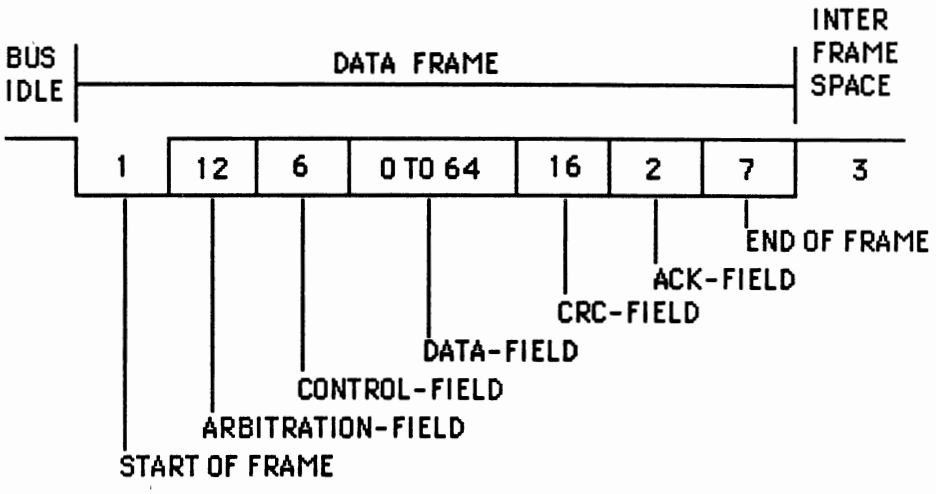


Figure 3. CAN Data Frame Format

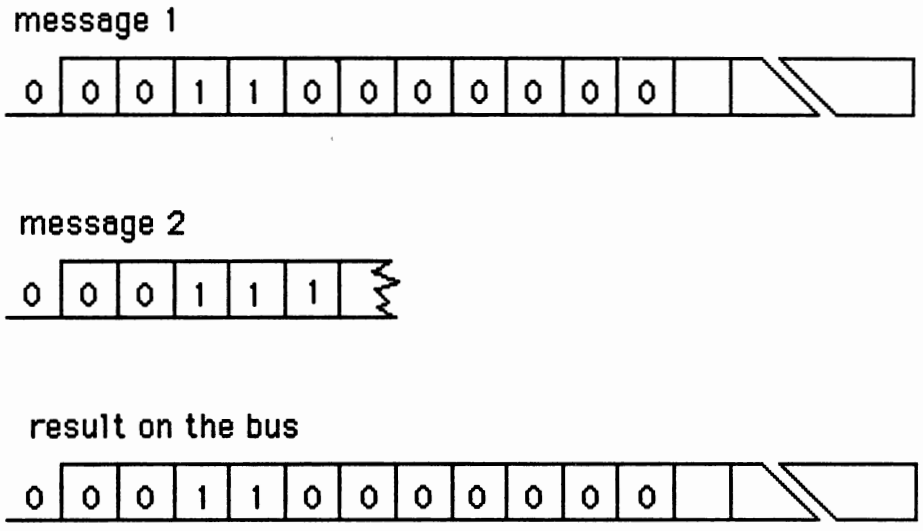


Figure 4. Example of Bitwise Arbitrition

collision will occur. As two message identifiers are sent to the bus bit by bit, the dominant 0 bit will overwrite the recessive bit so the identifier with the smaller binary value will overwrite the identifier with larger value. While a node is transmitting, it listens to see if the bit on the bus is the same as it is sending to the bus. If what it hears is different from what it sent, the node knows a collision has occurred and it will stop its transmission immediately. The higher priority message will continue until it is finished. After the higher priority message is finished, the lower priority message will try to transmit again. If there are other higher priority messages waiting to be transmitted, the lower priority message will lose arbitration again and again until all higher priority messages are finished. Figure 4 shows an example of the bit arbitration process. With this priority scheme, the delay of higher priority messages is reduced, and contention does not waste any bandwidth. Theoretically, it is possible for the channel utilization to reach 100%, if enough load is present.

The control field and data field follow the arbitration field. The control field consists of 6 bits, 2 reserved bits, and 4 data field length bits. The length of the data field is coded in bytes, and the control field identifies the number of bytes of data presented in the data field. The length of the data field varies from 0 to 8 bytes. The frame with a data field 0 is a special frame called a remote frame, which requests certain message to be sent to the bus. The contents of the arbitration field, control field, and data

field of the frame corresponds to the contents of the communication object to be transmitted or received in the DPRAM.

The CRC field contains a 15 bit cyclic redundancy check sum and a 1 bit delimiter. The check sum checks the star bit, arbitration field, control field, data field, and CRC field itself. After a message is completed the CRC field is checked. If any error is detected in the frame, the whole frame will be retransmitted.

The acknowledge field consists of two bits, the ACK_SLOT bit and the ACK_DELIMITER bit. The transmitting node sends the ACK_SLOT bit as 1. Any receiving node which has received a frame correctly will send a 0 to the bus at the same time. This 0 will overwrite the 1 sent by the transmitting node. Because the transmitting node listens to the bus, it will find the change and know that at least one station has received the message completely and correctly. The end of frame field consists of 7 bits, all of them 1. It marks the end of the frame.

Besides data frame and remote frame there are 2 more frames: The error frame, which indicates error conditions, and overload frame, which signifies receiving station not ready condition or wrong interframe space bit condition. Data frames and remote frames are preceded by at least 3 interframe spaces which allow the network interface to get ready to transmit or receive the next frame.

Assumptions and Conditions

Performance measure of local area network usually includes three parameters: throughput of the network, transmission delay, and communication channel utilization [stal90]. This chapter analyzes these parameters for the CAN.

In the following discussion it is assumed that the message generation occurs in a fashion which is characterized by the Poisson distribution, which is expressed as:

$$p(n) = \frac{e^{-\lambda} * (\lambda)^{n}}{n!}$$

In the above expression λ is the expected number of messages generated per second and p(n) is the probability of n message generations per second in the network. For certain applications the messages may be generated according to a certain pattern on some stations, but for right now we are not concerned with any particular application. In real situations message generation tends to approximate a Poisson distribution, and this is true especially when the number of stations and messages is large. In the same way the message length is also assumed to follow a Poisson distribution or exponential distribution.

With the above assumptions, the CAN is modeled as a single channel queuing system with Poisson arrival and negative exponential service time with the bus being the server and all waiting messages forming a single queue. There is no waste of bandwidth caused by collision like in CSMA/CD type networks, and no token passing time between data transmissions like in token bus and token ring networks.

The message transmission process is well described by the model. The priority scheme of CAN only influences the message order inside the queue. In the viewpoint of bus and the whole system the basic characteristics of the queue, such as expected number of messages in the system and probability of certain number of messages in the queue, are not influenced .

It is also assumed that all frames are data frames and remote frames because data frames and remote frames will dominate the network in normal situations and this assumption will simplify the analysis.

The possibility of message retransmission resulting from noise garbled frames is also ignored. This problem is related to the reliability problem which is usually analyzed in hardware design stage.

Some Conclusions from Queuing Theory

In order to do the following analysis, some conclusions from the queuing theory [Klei75, Mcmi73] need to be employed. These conclusions are summaried below. Let λ_i be the average message generation rate of i-th priority message, λ be the message generation rate of the whole system, and that there be totally n types of messages in the system. Then the message generation rate of the whole system is given by:

$$\lambda = \sum_{i=0}^{n-1} \lambda_i$$

In most distributed control systems, the length of each type of message is a constant. In CAN, each type of message

corresponds to a communication object in the DPRAM which is created at the system initiation stage. Thus, the length of each type of message is fixed. Let L_i be the length of i-th priority message and L be the expected message length of the whole system. Then L is given by:

$$L = \sum_{i=0}^{n-1} \frac{\lambda_i}{\lambda} * L_i$$

or

$$L = \frac{1}{\lambda} \sum_{i=0}^{n-1} \lambda_i * L_i$$

Let μ be the average number of messages the bus can transfer per second, B the bus bandwidth, OHF the overhead bits for each message, IFS the minimum interframe spaces between two consecutive frames, and F the expected message frame length. Then μ is given by:

$$\mu = \frac{B}{F + IFS}$$

and

$$F = L + OHF$$

The traffic intensity of the network, ρ, is given by

$$\rho = \frac{\lambda}{\mu}$$

When $\rho < 1$ the queuing system is in stable state and the following conclusions exist. Let P(n) represent the probability of n messages in the system, that is messages in the queue and on the bus, then P(n) is given by:

$$P(n) = (1 - \rho) * \rho^n$$

The probability of 0 message in the system is:

$$P(0) = 1 - \rho \ .$$

The expected number of messages in the system is given by

$$n_s = \frac{\lambda}{\mu - \lambda} \ .$$

The expected number of messages on the bus is given by

$$n_b = \frac{\lambda}{\mu} = \rho \ .$$

The expected number of message in the queue is given by

$$q = \frac{\lambda}{\mu - \lambda} - \frac{\lambda}{\mu} = \frac{\lambda^2}{(\mu - \lambda) * \mu}$$

Throughput of the Network

In this section the throughput characteristics of the CAN are studied. There are two kinds of throughput in the network: The effective throughput and throughput. Effective throughput, ES, is defined as the total number of data bits received at the destination stations correctly per second. Throughput of the network, S, is defined as the effective throughput plus overhead bits of each frame received at the destination stations correctly per second. Both the effective throughput and the throughput are a fraction of the bandwidth B. The load of the network is also divided into two kinds, the input load and offered load. The input load, IG, is defined as the total number of data bits generated per second by all stations in the network. The offered load, G, is the input load plus all overhead for sending these data. In some networks, the frames are very long, and the overhead

bits occupy just a very small portion of the total frame length. In that situation the overhead of each frame is ignored and the throughput and effective throughput are thought as the same. Input load and offered load are also thought as one. In CAN the maximum frame length is 108 bits and the overhead for each frame is 47 bits. The overhead of the network is not negligible. Let L be the expected length of the message, F the expected length of frames in the network, then the relationship among these parameters can be expressed as the following:

$$G = IG * \frac{F}{L}$$

$$S = ES * \frac{F}{L}$$

Let G_i and IG_i be the portion of the offered load and input load provided by i-th priority message, n the total number of messages in the network, then the offered load, G, and the input load of the system can be expressed as:

$$G = \sum_{i=0}^{n-1} G_i$$

$$IG = \sum_{i=0}^{n-1} IG_i$$

This network is in stable state when the offered load is smaller than the bandwidth, that is

$$G < B$$

In stable state there may be a waiting queue in the network and the number of messages waiting in the queue may change, but it is finite. All the message bits generated by all the stations in the network, plus necessary overhead bits, will be sent and received. The throughput of the network is:

$$S = G$$

and

$$ES = IG.$$

If the offered load is larger than the bandwidth, that is:

$$G > B$$

The throughput of the system can reach its maximum value:

$$S = B$$

and the effective throughput of the network will be:

$$ES = B * \frac{L}{F}$$

and

$$ES <> IG$$

Figure 5 shows the relationship between the throughput and the offered load given by the above formulae. Both the throughput and the offered load are expressed as a fraction of the bandwidth.

According to the CAN protocol, when the offered load is larger than the bandwidth, the higher priority messages can still be fully transmitted, but lower priority messages may have no chance to be transmitted or can only be partly transmitted depending on how much bandwidth has been left for this type of message. The bandwidth used by higher priority messages is:

$$\sum_{k=0}^{i-1} G_k$$

In the above expression, the highest priority message is defined to have priority 0. This is the way message priority is defined in CAN. The remaining portion of the bandwidth, which can be used by the i-th priority message and lower

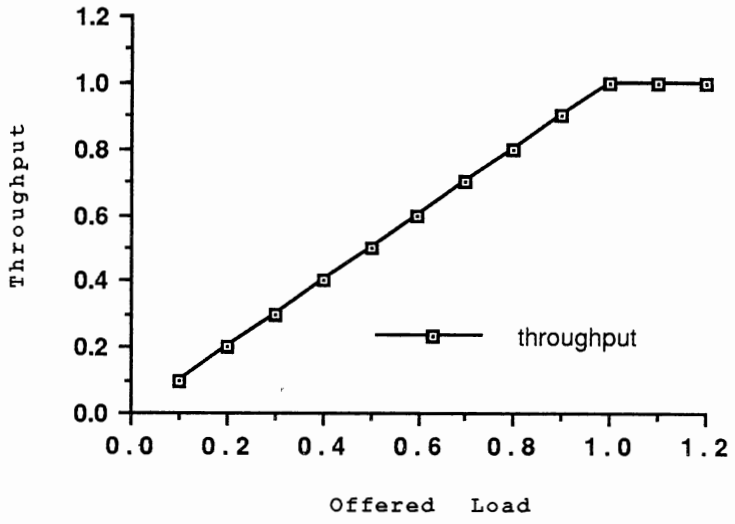


Figure 5. Theoritical Relationship of Offered Load and Throughput

priority messages, is:

$$B - \sum_{k=0}^{i-1} G_k$$

If the remaining portion of the bandwidth is larger than the offered load of the i-th priority message, then the throughput of i-th priority message is equal to its offered load. Otherwise throughput of the i-th priority message can only be the remaining part of the bandwidth. It is possible that the above expression gives a negative value, but throughput of any message can never be negative. Let S_i be the throughput of i-th priority message, which is the portion of data bits and overhead bits received by the destination station(s), and belong to i-th priority message. Then the throughput of the i-th priority message is given by

$$S_i = \max(\min(G_i , \quad B - \sum_{k=0}^{i-1} G_k) , \quad 0)$$

Let L_{max} be the maximum length of the data field(64 bits), F_{max} the maximum frame length(108 bits), and IFS the minimum number of interframe spaces required between frames(3 bits). When there are enough loads present and the data field of all messages is maximum length, the effective throughput of the network can reach its maximum value, which is:

$$ES_{max} = \frac{L_{max}}{F_{max} + IFS} * B$$

If the offered load is equal to the bandwidth then this is also the maximum input load.

Delay of Different Priority Messages

In CAN each type of message has a unique identifier that represents a unique priority and each type of message corresponds to a unique communication object in a certain CAN node, so at any time there couldn't be more than one certain type of message waiting for transmission. In most distributed control systems all messages are pre-defined and the length of each type of message is a constant. In CAN, when a station is powered up the station CPU runs a program to initiate the dual port RAM(DPRAM) on the node and creates data structures for communication objects, thus defining the length of each type of message

The delay of a message is defined as the time interval between the time a message is generated and the time this message is received by the destination station(s). According to the CAN protocol, delay of certain type of messages is influenced only by higher priority messages. Lower priority messages have no influence on higher priority messages.

Maximum Delay

Basically, all other messages have no influence on the delay of the highest priority messages because whether there are other messages waiting for transmitting or not once the bus becomes available, these messages will begin to transmit until the transmission finished. The maximum delay of the highest priority message appears when this message generates just after the longest message has transferred its first bit and seized the bus. This highest priority message has to

wait until the current message finishes its transmission. So the delay of this highest priority message is the time the current message needs to finish its transmission plus the time for interframe spaces and the time needed to transmit this message itself. The longest message is 108 bits(1 bit of start of frame, 12 bits of arbitration field, 6 bits of control field, 64 bits of data field, 16 bits of CRC field, 2 bits of acknowledge field and 7 bits of end of frame field) and there are at least 3 interframe spaces between data frames. Let D_0 be the maximum delay of the highest priority message, F_0 be the frame length for the highest priority message, F_{max} the length of the longest frame(108 bits), IFS the minimum number of interframe spaces between frames(3 bits), and let the time unit be the time needed to transfer 1 bit (1/B second), then the maximum delay of the highest priority message is:

$$D_0 = F_{max} + IFS + F_0 - 1$$

The maximum delay of i-th priority message appears when this message is generated with all higher priority messages at the same time, and at that moment a longest message has just transferred one bit and seized the bus; in the waiting time, higher priority messages continue to generate at their maximum rate. Let the maximum delay of i-th priority message be the maximum waiting time of this message plus the time needed to transfer this i-th priority message itself. The maximum waiting time is the time needed to finish the current message on the bus plus the time needed to transfer all higher priority message waiting in the queue and arriving during the waiting time. Let D_i be the maximum delay of

i-th priority message, W_i the maximum waiting time for i-th priority message, F_k the length of the frame for k-th priority message, and M_k the maximum message generate rate of k-th priority message, then the maximum waiting time is given by the following equation:

$$W_i=(F_{max}+IFS-1)+\sum_{k=0}^{i-1}(F_k+IFS)+\sum_{k=0}^{i-1}(F_k+IFS)*M_k*\frac{W_i}{B}$$

Solving the equation for W_i gives:

$$W_i = \frac{(F_{max}+ IFS - 1) + \sum_{k=0}^{i-1}(F_k + IFS)}{1 - \frac{\sum_{k=0}^{i-1}(F_k + IFS) * M_k}{B}}$$

Maximum delay of the k-th priority message is equal to the maximum waiting time plus the time needed to transfer the message itself. That is:

$$D_i= W_i+ F_i.$$

or

$$D_i = \frac{(F_{max}+ IFS - 1) + \sum_{k=0}^{i-1}(F_k + IFS)}{1 - \frac{\sum_{k=0}^{i-1}(F_k + IFS) * M_k}{B}} + F_i$$

Figure 6 shows the relationship between maximum delay and message priorities and figure 7 shows the relationship between maximum delay and offered load. Under light or medium load, message priorities do not have much influence on the maximum delay, but when the offered load reaches about 80

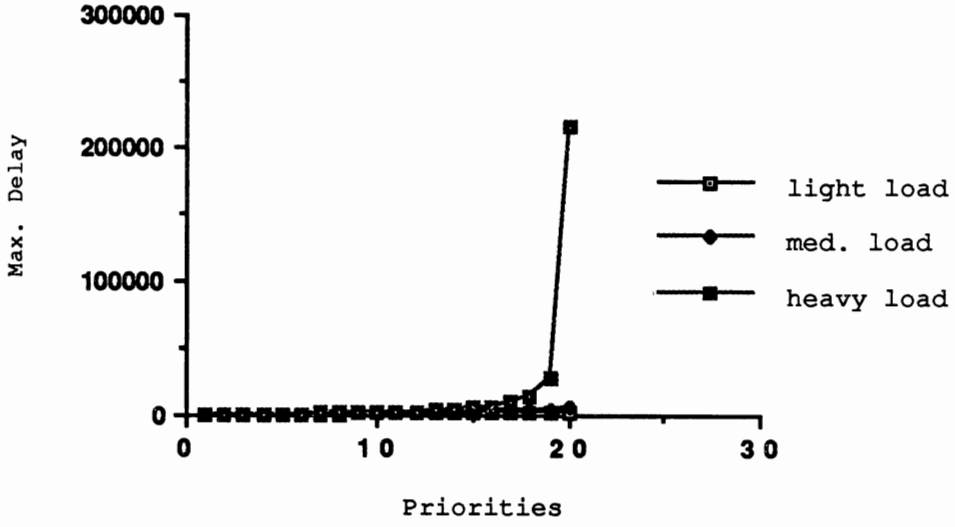


Figure 6. Theoretical Relationship between Max. Delay and Message Priorities

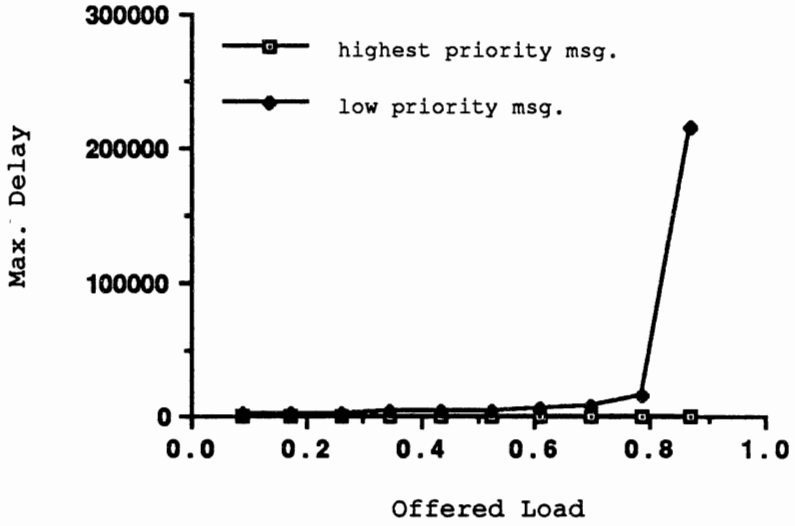


Figure 7. Theoretical Relationship between Max. Delay and Offered Load

percent of the bandwidth the maximum delay of lower priority messages begin to increase sharply. Maximum delay of higher priority messages is not influenced by the offered load.

Expected Delay

In the following analysis it is assumed that the network is in stable state, that is $\rho < 1$, so that all conclusion from the queuing theory can be employed for the analysis. In general a highest priority message may be generated at any time, that is, when it is generated there may or may not be a message on the bus. If there is no message on the bus then the generated message will be transmitted immediately. If there is a message on the bus then it may be any type of message belonging to this system. To find out the average delay of the highest priority message we can use the expected value of message frame length, F, to represent the length of different type of message frames. It is also reasonable to assume that the probability of 1 bit, 2 bits, . . . , up to F bits of the current message or any one of the 3 interframe spaces has been transmitted are equal, that is the probability is:

$$\frac{1}{F + IFS}$$

The expected number of bits already transmitted is:

$$\sum_{i=1}^{F+IFS} \frac{1}{F+IFS} * i = \frac{1}{F+IFS} * \frac{F+IFS+1}{2} * (F+IFS) = \frac{F+IFS+1}{2}$$

thus the number of bits including the interframe spaces remaining to be transmitted is:

$$(F + IFS) - \frac{F + IFS + 1}{2} = \frac{F + IFS - 1}{2} .$$

So the expected waiting time of the highest priority is:

$$w_0 = 0*P(0) + (1-P(0)) * \frac{F + IFS - 1}{2}$$

or

$$w_0 = \rho * \frac{F + IFS - 1}{2}$$

which is the product of the probability of one or more messages in the communication system and the expected number of bits remaining to be transmitted. In the above two formulas $P(0)$ is the probability of no message in the network, $1 - P(0)$ is the probability of one or more messages in the network. From queuing theory we know that

$$1 - P(0) = \rho$$

and the first formula for w_0 is just standard expression for expected value of something. The expected delay of the highest priority message is the sum of the expected waiting time and the time needed for transmitting the highest priority message itself, that is:

$$d_0 = w_0 + F_0$$

or:

$$d_0 = \rho * \frac{F + IFS - 1}{2} + F_0$$

The expected delay of the i-th priority message should be the sum of the expected waiting time of the i-th priority message and the time needed for this message to be transmitted.

The expected waiting time for the i-th priority message consists of three parts: the expected time for current

message on the bus to finish; the expected time for transferring higher priority message waiting in the queue, and the expected time for transferring the higher priority messages generated during the waiting time.

The waiting time needed for the current message on the bus to finish should be the product of the expected number of messages on the bus and the expected number of bits remaining to be transmitted in the current message. From queuing theory, the expected number of messages on the bus is equal to ρ and the expected number of bits is the same as in the case of the highest priority message, that is:

$$\frac{F + IFS - 1}{2}$$

So the expected waiting time needed for the current message to finish is:

$$\rho * \frac{F + IFS - 1}{2}$$

The second part of the waiting time is the time needed to transmit the higher priority messages waiting in the queue. According to the queuing theory the expected number of message waiting in the queue is:

$$q = \frac{\lambda^2}{\mu(\mu-\lambda)}$$

Among these messages, only those messages with higher priority will cause the i-th priority message to wait. The number of these higher priority messages should be proportional to the number of messages they generate per second, so the expected number of higher priority messages is given by:

$$q_h = q * \frac{\sum_{k=0}^{i-1} \lambda_k}{\lambda}$$

or

$$q_h = \frac{\lambda * \sum_{k=0}^{i-1} \lambda_k}{\mu * (\mu - \lambda)} .$$

and the expected time needed to transfer the higher priority messages waiting in the queue is:

$$q_h * F = \frac{\lambda * \sum_{k=0}^{i-1} \lambda_k}{\mu * (\mu - \lambda)} * F$$

The third part of the waiting time is the time needed to transmit higher priority messages generated during the waiting time. Let w_i be the expected time the i-th priority message has to wait, F_k the length of the frame for the k-th priority message which is (L_k+OHF), then the time needed for i-th priority message to wait for the incoming higher message to be transmitted during the waiting time is:

$$\sum_{k=0}^{i-1} (F_k + IFS) * \lambda_k * \frac{w_i}{B}$$

The total expected waiting time for the i-th priority message is given by the following equation:

$$w_i = \rho * \frac{F + IFS - 1}{2} + \frac{\lambda * \sum_{k=0}^{i-1} \lambda_k}{\mu (\mu - \lambda)} * F + \sum_{k=0}^{i-1} (F_k + IFS) * \lambda_k * \frac{w_i}{B}$$

Solving the equation for w_i gives:

$$w_i = \frac{\rho * \frac{F+IFS-1}{2} + \frac{\lambda * \sum_{k=0}^{i-1} \lambda_k}{\mu(\mu-\lambda)} * F}{1 - \frac{\sum_{k=0}^{i-1} (F_k+IFS) * \lambda_k}{B}}$$

The expected delay of i-th priority message is equal to the expected waiting time plus the time needed to transmit this message itself, that is:

$$d_i = w_i + F_i$$

or

$$d_i = \frac{\rho * \frac{F+IFS-1}{2} + \frac{\lambda * \sum_{k=0}^{i-1} \lambda_k}{\mu(\mu-\lambda)} * F}{1 - \frac{\sum_{k=0}^{i-1} (F_k+IFS) * \lambda_k}{B}} + F_i$$

In the above analysis, the expected number of messages in the system is not used to find the waiting time. Instead, the expected number of messages in the queue and the expected number of messages on the bus were used. This is because the current message on the bus may be a message with lower priority.

Figure 8 shows the relationship between expected delay and message priorities and figure 9 shows the relationship between expected delay and offered load. When offered load is increased to about 60 percent of the bandwidth the expected delay of lower priority message begins to increase,

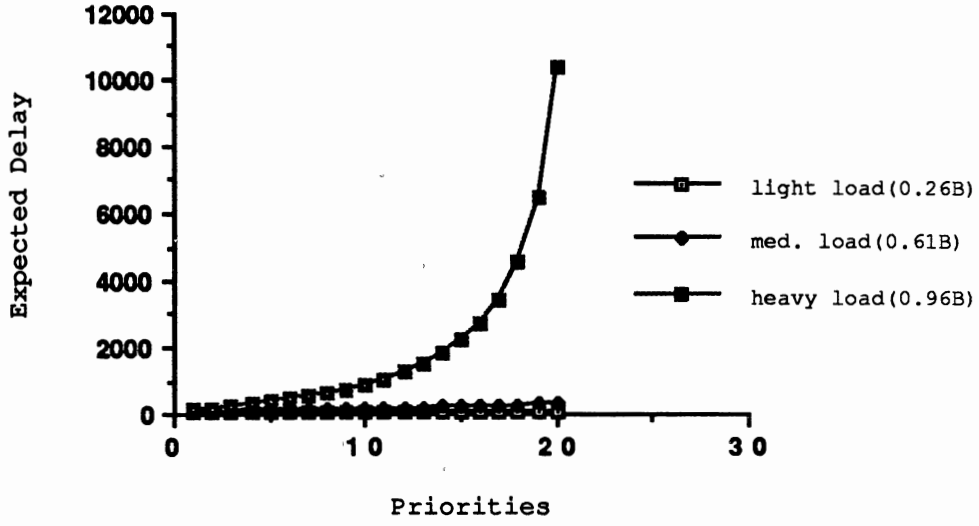


Figure 9. Theoretical Relationship between Expected Delay and Offered Load

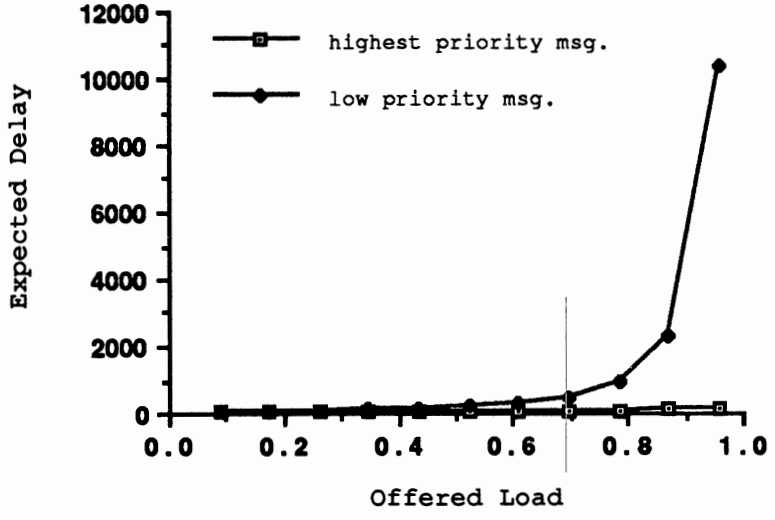


Figure 8. Theoretical relationship between Expected Delay and Message Priorities.

but not as sharply as the maximum delay increases.

Communication Channel Utilization

Communication channel utilization is defined as the fraction of the total channel capacity being used. In the above analysis if throughput of the network is normalized and expressed as a fraction of channel capacity rather than number of messages or number of data bits and overhead bits been transferred then it can be interpreted as channel utilization. Let S be the throughput of the system, U the channel utilization and B bandwidth of the network, then the channel utilization of the network is:

$$U = \frac{S}{B}$$

CHAPTER IV

MESSAGE PRIORITY ASSIGNMENT ALGORITHM

The Need for the Algorithm

All messages in CAN have to be assigned a priority at the network design stage. In real-time distributed control systems message transmission must be in time. For every message there is an allowed maximum delay within which this message must be transmitted and received by the destination station after it is generated. Even in the worst case situation this limitation should not be exceeded. There are many different types of messages and some of them are much more important than others and must be transmitted and received within much shorter time. The purpose of the priority scheme in CAN is to guarantee that important messages are transmitted earlier than less important ones. To guarantee that the allowed maximum delay requirement of every message in the real-time system is satisfied, the priority assignment has to be based on the worst case situation. Let D_i be the maximum possible delay of i-th priority message, and AD_i be the allowed maximum delay of that message, then the condition

$$D_i <= AD_i$$

must be satisfied for all messages in the system.

The most straightforward way of assigning priorities to

messages is according to the importance of each type of message, which is expressed as the allowed maximum delay of each type of message. This method of assigning priorities does not generate an optimum assignment and may give an incorrect assignment when the given set of requirements are conflicting. Now let's define the term "optimum" for the message priority assignment. From the whole system's viewpoint, an optimum assignment scheme should make the average delay of all messages minimal. This problem is similar to the job scheduling problem in operating system design in which the "shortest job first" is the solution [Deit90]. For the CAN there are extra constraints, that is the allowed maximum delay constraints, so the optimum is a kind of conditional optimum or optimum with certain constraints. The idea of the "short job first" can be illustrated by the following example.

Three messages, with transmission times of 10, 100, and 1000 units respectively, arrive at the transmission channel at the same time. The channel can transmit just one message at a time. There are several ways to arrange the transmission and different arrangements yield different average delay or total delay of these three messages. Let t_i represent the transmission time of number i transmitted message, d_i represent the delay of this number i transmitted message, D represent the total delay of these three messages, and AVD represent the average delay of these three messages then

$$d_i = \sum_{j=1}^{i} t_j$$

$$D = \sum_{i=1}^{3} d_i = \sum_{i=1}^{3} \sum_{j=1}^{i} t_j$$

$$AVD = \frac{D}{3}$$

For these three messages there are six different transmission arrangements totally. Table I lists three transmission arrangements and the resulting delay of each message, total delay and average delay of all three messages. Arrangement #1 transmits the shortest message first and the longest message last and the resulted average delay is 410 time units. Arrangement #3 transmits the longest message first and the shortest message last, with an average delay of 1070 time units. From the example it can be seen that transmitting shorter messages earlier or, "short job first", makes the total delay and average delay of all the messages minimal.

According to this principle, higher priorities should be assigned to messages which are transmitted less frequently and have shorter message length. In other words, higher priorities should be given to messages that use the transmission channel less.

So the solution for CAN should be one which can satisfy the allowed delay requirement of all messages and make the average delay of all messages reach the lowest possible value. The following priority assignment algorithm is based on the above idea. First the priorities are assigned to the

messages according to the "short job first" principle, then adjustments are made to make priorities of all messages satisfy the allowed delay requirement. Because the algorithm is based on the worst case and the probability of the worst case happening is very low, the adjustment makes the allowed delay of each message merely satisfied to make the assignment scheme as optimum as possible. It can be proved that such a solution is unique if it does exist.

TABLE I

COMPARISON OF DIFFERENT MESSAGE TRANSMISSION ARRANGEMENTS

arrange	t_1	t_2	t_3	d_1	d_2	d_3	TD	AVD
#1	10	100	1000	10	110	1110	1230	410
#2	10	1000	100	10	1010	1110	2130	710
#3	1000	100	10	1000	1100	1110	3210	1070

Another function of this algorithm is that through the systematic search a set of incompatible requirements can be detected in which the control system designer should reconsider his or her design and the requirement on the network. For example, consider a system that has two

messages, each having a transmission time of 100 units and an allowed maximum delay of 150 units. There is no way to satisfy the requirements of these two messages, because if both of them arrive at the transmission channel at the same time one of them has to wait 100 units; its delay would be 200 units, which is larger than its allowed maximum delay. For a given set of requirements, determining whether they are compatible and all can be satisfied is not an easy question, especially when the message set is big. Only a systematic search can give an accurate answer. Simulation can hardly work. Also, this problem must be considered in the CAN design process. If one works on a set of incompatible requirements blindly he or she will certainly waste time and will fail in the end.

Two Examples

The algorithm can be illustrated best by the following simplified examples. Table II gives ten types of messages in a control system. For each type of message there is a message name, a transmission time which is the time need for transferring this message, and an allowed maximum delay. To simplify the example, the arrival of higher priority messages during the waiting time is omitted. Priorities are assigned to each of these messages so that when two or more messages arrive at the transmission channel at the same time, the higher priority message is transmitted before the lower priority messages. For every message the priority assignment should make the maximum possible delay of that message less than or equal to its allowed maximum delay. The maximum

TABLE II

TEN MESSAGES OF A CONTROL SYSTEM

Message Name	Transmission Time	Allowed Maximum Delay
A	100	500
B	500	2400
C	50	850
D	600	3100
E	700	1900
F	50	300
G	200	800
H	400	1200
I	300	3500
J	50	350

delay appears in the worst case of a message transmission, in which a message arrives at the communication channel with all higher priority messages at the same time and must wait for all higher priority messages to finish. Besides, the assignment should make the average of the maximum delay of all messages or the sum of the maximum delay of all messages as small as possible. Let t_i be the transmission time of i-th priority message, D_i the maximum delay of i-th priority message in the worst case, TD the sum of the maximum delay of all messages, and AD_i the allowed delay of i-th priority message the above condition can be expressed as

$$D_i = \sum_{j=1}^{i} t_j$$

$$D_i \quad <= \quad AD_i \qquad (5.1)$$

$$TD = \sum_{i=1}^{10} D_i = \sum_{i=1}^{10} \sum_{j=1}^{i} t_j$$

The algorithm first sorts the messages and then puts them in a list according to their transmission time. The message with the smallest transmission time is put on the leftmost side of the list and named message number 1. The following messages are named number 2, number 3, through number 10. This is illustrated on the first row of figure 10. Each type of message is represented by a rectangle with the message number on the top of the rectangle, the transmission time in the middle, and the allowed maximum delay on the bottom of the rectangle. The position of each message in the row represents the priority of each message with the leftmost position representing the highest priority. Changing

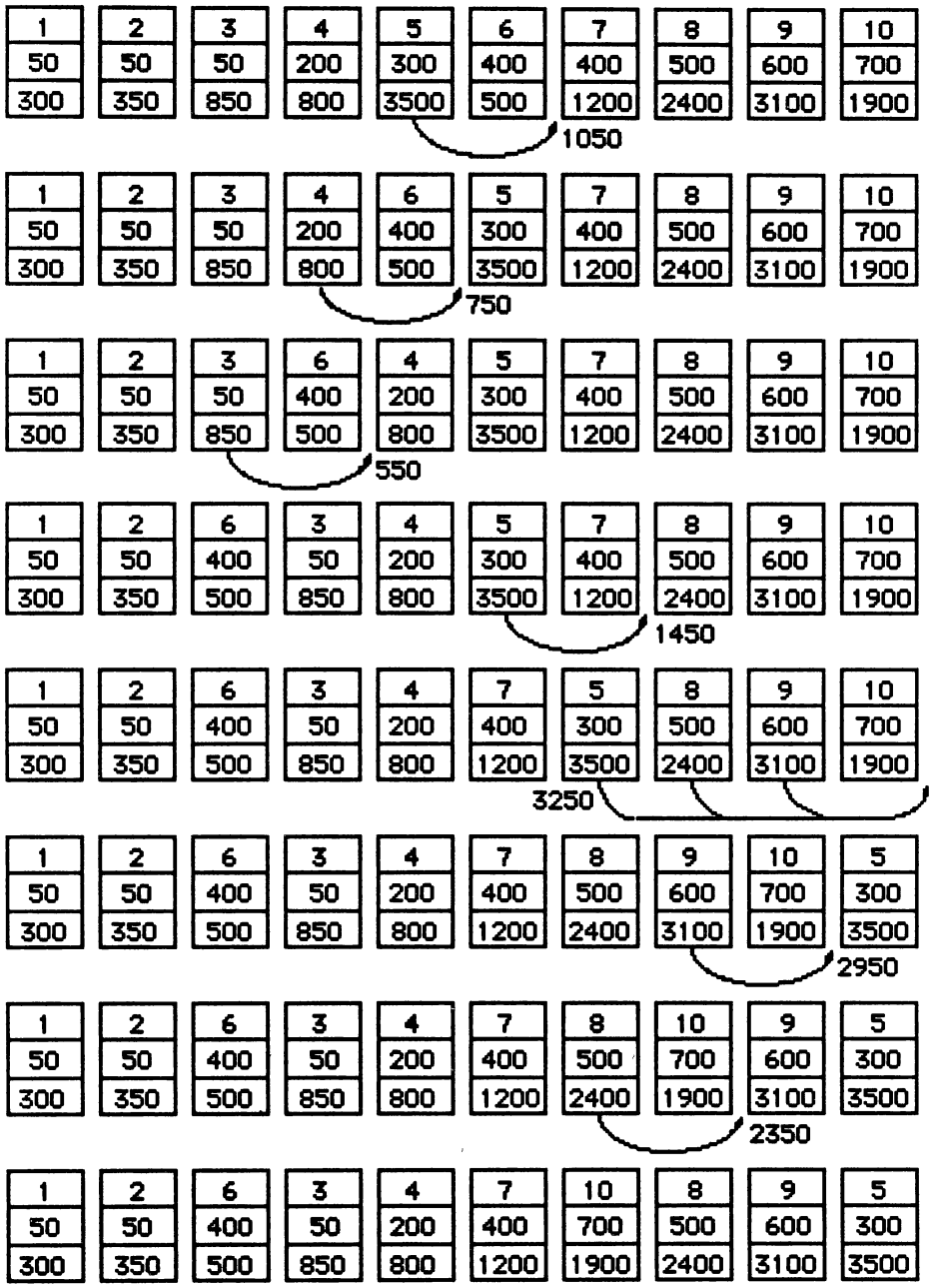


Figure 10. Example 1 of Prioroty Assignment Algorithm

position of a message means changing the priority of that message, so the list represents a priority assignment scheme. The first row of figure 10 represents the priority assignment using the short job first method. It has the minimal total delay TD, but whether the worst case delay of each message exceeds its allowed maximum delay or not is unknown. That is, the condition

$$D_i <= AD_i$$

for every message is unknown. So the next step is to check this condition for every message. The check begins from message number 1 to number 10, one at a time. The position of a message may be changed but its number will never be changed. This way it can be sure that no message will be ignored.

On the first row of figure 10, message number 1, 2. 3, 5, and 6 are checked. For message 1 the maximum possible delay or worst case delay and the allowed maximum delay are

$$D_1 = 50 ,$$

$$AD_1 = 300 ,$$

$$D_1 < AD_1,$$

condition (5. 1) is satisfied. For message 2, The worst case delay and allowed maximum delay are

$$D_2 = 100,$$

$$AD_2 = 350,$$

condition (5. 1) also satisfied. For message 3, 4, and 5, condition (5. 1) are all satisfied. For message 6,

$$D_6 = 50 + 50 + 50 + 200 + 300 + 400$$

$$= 1050,$$

$AD_6 = 500,$

$D_6 > AD_6,$

condition (5. 1) is not satisfied. In this situation the algorithm makes some adjustments on the priority assignment. It tries to reduce the worst case delay of the current message by selecting and moving messages from the left of the current message to the right of the current message. Messages on the left side of the current message have all been checked and all satisfy condition (5. 1). The movement should not disturb the existing satisfaction of condition(5. 1). Now the algorithm tries to shift message 6 left one position and move message 5 after message 6. The priorities of message 5 and 6 are exchanged. In the new scheme

$$D_6 = 50 + 50 + 50 + 200 + 300 + 400$$
$$= 1050$$
$$AD_6 = 3200$$

condition (5. 1) is satisfied, and this adjustment can be made. So the algorithm made this adjustment and the new priority assignment scheme is the second row of figure 10. In the new position message 6 still cannot satisfy condition (5. 1), so the above process repeats again until message 4 and 3 have been moved to the right of message 6. The result is the 4th row of figure 10. Now the algorithm finishes checking message 6, the next message checked is message 7. It is shifted left one position and message 5 is moved to its right. The result is the 5th row of figure 10. Message number 8 and 9 satisfy condition (5. 1) in their original position and adjustment is not needed. For message 10

$$D_{10} = 50 + 50 + 400 + 50 + 200 + 400 + 300 + 500 + 600 + 700$$

$$= 3250$$

$$AD_{10} = 1900$$

condition (5. 1) is not satisfied, so some adjustments have to be made. First the algorithm tries to move message 9 after message 10. If this adjustment is made, in the new scheme

$$D_{10} = 50 + 50 + 400 + 50 + 200 + 400 + 300 + 500 + 700$$

$$= 3250$$

$$AD_{10} = 3100$$

condition (5. 1) is not satisfied and this adjustment cannot be made. In this situation the algorithm attempts to shift messages 9 and 10 one position to the left and move message 8 after message 10. If this movement were made then in the new scheme

$$D_{10} = 50 + 50 + 400 + 50 + 200 + 400 + 300 + 600 + 700 + 500$$

$$= 3250$$

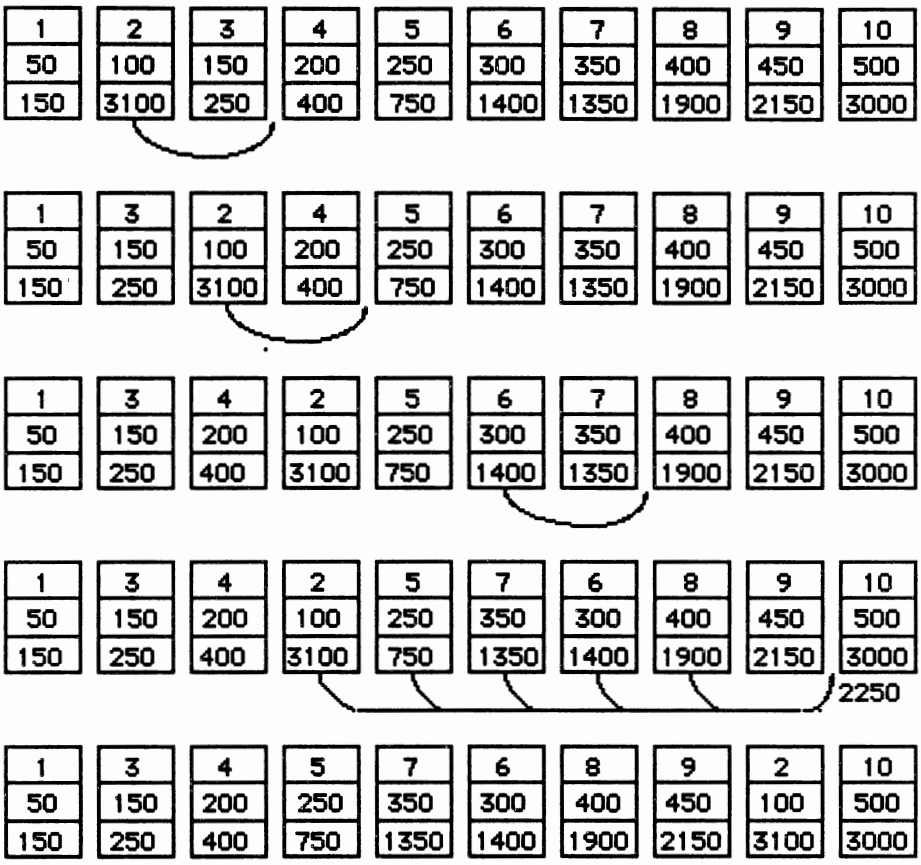
$$AD_{10} = 2400$$

condition (5. 1) is still not satisfied, and this adjustment cannot be made. Again the algorithm shifts left one position to select message 5 and try to shift messages 8, 9, and 10 left one position and put message number 5 after message 10. This time the condition is satisfied, and the result is row 6 of figure 10. The algorithm still needs to check to ascertain that the priority of message 10 can satisfy condition (5. 1) in the new position. If the condition

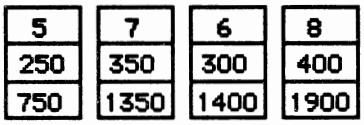
cannot be satisfied the above process has to be repeated again and again until message 10 has been moved to a position in which condition (5. 1) could be satisfied. So the process has to repeat two more times, and message 10 is moved two positions left and messages 9 and 8 are moved after message 10 again. The result is row 7 and row 8 of figure 10, and row 8 is the final result.

In general, when a message cannot satisfy condition (5. 1) the algorithm tries to move some messages from the left of this message to the right of this message, one at a time. The algorithm selects messages from right to left, beginning with the left neighbor of the current message. If the message on the leftmost position has been selected and failed, there is no answer to this set of requirements. No message priority arrangement can satisfy the allowed maximum delay requirement for this set of messages in the worst case situation.

Figure 11 shows another example in which message 7 was shifted one position left and message 6 was moved to position 7 to make message 7 satisfy condition (5. 1). This is shown on row 3 of figure 11. Later messages 5, 7, 6, 8, and 9 were shifted left one position and message 2 on position 4 was moved to position 9 to make message 9 satisfy condition (5. 1). Because message 2 was moved from the left of messages 5, 7, 6, and 8, their worst case delays are all decreased, and previous adjustment of message priorities inside this sublist becomes unnecessary, thus making the assignment less optimum. To keep the priority assignment



Apply algorithm to sublist



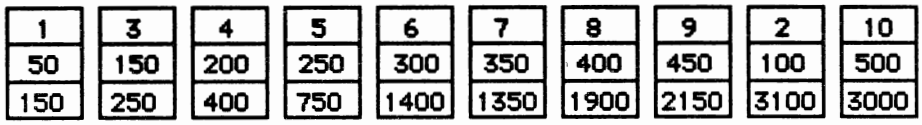


Figure 11. Example 2 of Priority Assignment Algorithm

scheme as optimum as possible the algorithm should be applied to the sublist again. In the example, message 7 was moved back to right of message 6 again.

The Algorithm

The following is the formal description of the priority assignment algorithm. Suppose there are N messages in the system and for each message there is a transmission time t and an allowed maximum delay AD associated with it. In the description, subscripts i, p, and q identify the position or priority of a message, not the message number. For example, t_i means transmission time of the message on position i, not the transmission time of message i. Message numbers are used to make sure that every message is checked. Variable k is used to store the message number.

1. Sort all the messages according to their transmission time t and put them in a list in ascending order. If 2 or more messages have the same transmission time t, then arrange them according to their allowed maximum delay AD.
2. Number all messages from 1 to N in the above order. These numbers is permanent names for these messages.
3. Assign priority to all messages according to their position in the above list with 1 as the highest priority. Later the positions of these messages may be changed. Changing position of a message means to change the priority of that message.

4. k <- 0.

5. k <- k + 1

   If k = N + 1 then a solution has been found, stop.

6. Assign position of message k to variable p and q.

7. If the allowed maximum delay of the message on position p is satisfied, that is,

$$D_p = \frac{(FL_{max}+IFS-1) + \sum_{j=0}^{p-1} (F_j+IFS)}{1 - \frac{\sum_{j=0}^{p-1} (F_j + IFS) * M_j}{B}} + F_p < AD_p \qquad (5.1)$$

   then go to step 5.

8. q <- (q-1)

   If q=0 then no solution exists. stop.

9. If the message on position q cannot be moved right after the message on position p, that is, after shifting messages on position q+1,q+2 . . . , p one position left and moving the message on position q to position p, condition (5. 1) cannot be satisfied, then go to step 8.

10. Shift messages on position q+1, q+2, . . . , p one position left and move the message on position q to position p.

11. If (p - q) >= 3 then apply the algorithm to sublist form position q to position (p - 2). All messages on the left of position q is treated as one big message.

12. p,q <-(p - 1); go to step 7.

Proof of the Algorithm

This section proves that the above algorithm generates an optimum message priority assignment scheme. That is, the resulting assignment is one which can satisfy the delay requirement of all messages and make the average of total delay of all messages reach the lowest possible value.

Here an assignment is still represented by a list of messages. The position of a message in the list still represents the priority of a message and "message i" means i-th priority message. For an assignment generated by the above algorithm, suppose the position of message u and v has been exchanged and $u < v$. There are three situations:

1. $t_u < t_v$

Originally the waiting time of message u+1 is

$$w_{u+1} = \sum_{i=1}^{u} t_i$$

After the exchange the new t_u has increased, so the waiting time of message u+1, w_{u+1} is increased. Similarly, the waiting time of message u+2, u+3, . . . , v, w_{u+2}, w_{u+3}, . . . ,w_v is also increased. The waiting time of message 1, 2, . . . ,u and message v, v+1, . . . , n is not influenced by the exchange, so the total waiting time of all messages increased. The total delay is given by

$$D = \sum_{i=1}^{n} d_i$$

$$= \sum_{i=1}^{n} (w_i + t_i)$$

$$= \sum_{i}^{n} w_i + \sum_{i}^{n} t_i$$

In the above formula the summation of message transmission time is a constant; it is not influenced by any arrangement. When total waiting time increased the total delay is increased. Thus the average delay increased.

2 $t_u > t_v$

According to the algorithm a message with larger t is moved left over messages with small t only for satisfying the allowed maximum delay requirement of that message, and it is moved to a position which can barely satisfy the allowed delay requirement of that message. This is done in step 7, 8, 9, 10 and 11 of the algorithm. Because $t_u > t_v$, message u must be moved by the algorithm to satisfy the delay requirement. In the original list, message u is in a position which can barely satisfy the allowed maximum delay requirement. Exchanging the positions of message u and message v moves message u right; then the priority assignment of message u is no longer satisfy the delay requirement. So such exchange cannot be made.

3. $t_u = t_v$

In this situation, the exchange of the position of message u and message v may or may not violate the allowed maximum delay requirement of message u, but it has no influence on the total delay D or average delay of all messages in the system.

From 1, 2, and 3, it can be said that any other arrangement of the priority cannot be better than the one generated by

the above algorithm. Any change to the assignment generated by the algorithm are either violate the allowed maximum delay requirement of some message or increase the total delay of all messages in the system or has no effect on the total delay of all the messages in the system. So the priority assignment generated by the above algorithm is the only one which can satisfy the allowed maximum delay requirement of all messages and make the total delay of all messages minimal, thus is an optimum one.

CHAPTER V

SIMULATION OF THE CAN

Description of the Simulation Program

Both to verify the analytical model and to provide a tool for the network design a simulation program was developed. This simulation program is written in Turbo C and runs on IBM PC and compatible computers; hence it is very easy to find a computer to run it.

In the simulation program the bus is represented by an integer variable and each station is represented by a structure. Communication processes are simulated by the simulation program as a series of message cycles which, in turn, consists of a series of bit cycles. A message cycle corresponds to the transmission of one message and a bit cycle corresponds to the transmission of one bit. There are a set of flags to signify various status, for example whether a station has messages to transfer, or what field the current bit on the bus comes from. Message arbitration is performed in the transmission and receiving of message ID. The function

$$F(x) = - (\frac{B}{\lambda} * \ln x)$$

is used to generate exponential message inter-arrival time, so the message generation will follow Poisson distribution.

In the above formula F(x) is the exponential message inter-arrival time. X is the uniform distributed random number between 0 and 1. B is the bandwidth of the communication channel and λ is the expected message arrival rate of the communication system. CRC is not simulated because there is no any errors in the simulated message transmission. Also, in the previous analysis the possibility of transmission error is ignored.

To run the simulation program the user needs to specify the whole network to the simulation program: what is the bandwidth of the bus, how many stations are on the bus, how many types of messages are transmitted for each station, the message ID , message length, and message transmission rate for each type of transmitting message, how many types of messages a station receives and the message ID of each type of receiving message.

The simulation program is capable of producing the following performance results: the bus busy time measured in bit time, the bus idle time, bus utilization, total collision experienced in the whole process, total number of messages generated, total number of message transmissions started, total number of message transmissions completed, maximum queue length, number of messages transmitted for each type of message, average delay and maximum delay of each type of message. The user can watch the whole simulation process and stop it at any time to check the correctness. If there is enough disc space the whole simulation process can be recorded to a file for output. The only limitation for the simulation program is that the system clock cannot go beyond

the maximum real value the machine can express.

The simulation program may be used as a design tool. A network design can be quickly evaluated by running the simulation program using the design parameters. The simulation program also checks possible design problems for the users. For example two stations may use the same message ID for their transmitting messages, a station may transmit a type of message without a station receiving it or a station may be specified to receive a type of message without a station sending it.

Simulation Results

This section presents the performance characteristics of the CAN obtained via the simulation. Twelve sets of data are used to run the simulation program. In order to be compatible, every set of data includes five stations, every station transmits and receives four types of messages, every message is five bytes long, and only the message transmission rate of each message changes to make the offered load of the system change from approximately 10% of the bandwidth to 100% of the bandwidth. Twenty priorities are assigned to this twenty types of messages randomly. In fact the same sets of data have been used to produce the figures in chapter four using the derived formulae.

Figure 12 shows the throughput vs. offered load characteristics. The throughput is expressed as a fraction of the bandwidth. From figure 12 it can be seen that the relationship between the offered load and the throughput is

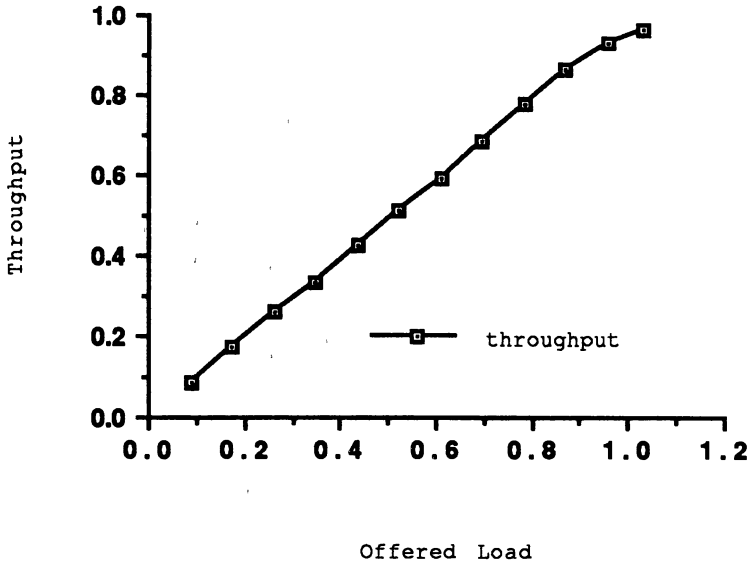


Figure 12. Throughput of the System vs. Offered Load

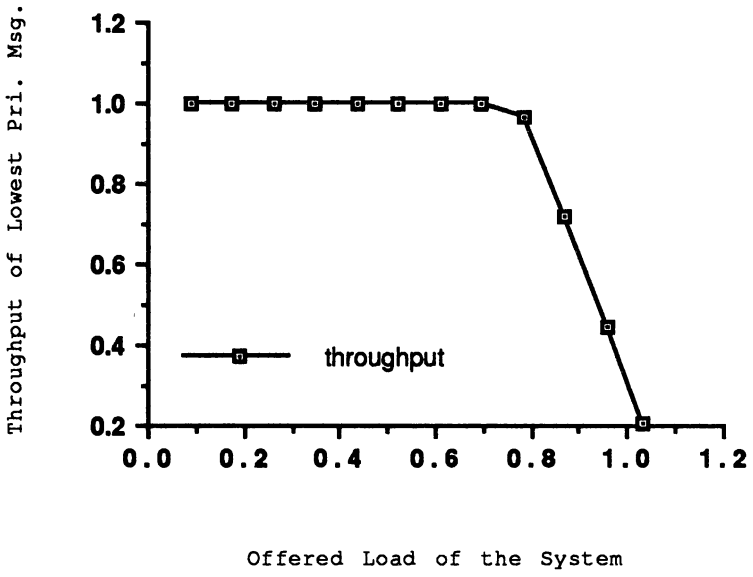


Figure 13. Throughput of the Lowest Priority Message vs. Offered Load of the System

almost linear, that is the throughput increases with the offered load in a constant ratio, and it agrees with the relationship derived in chapter four. Figure 13 shows the throughput of the lowest priority message vs. the offered load of the network. When the offered load of the network is not very heavy the throughput of the lowest priority message is not influenced by the increase of the offered load of the network. All messages of this type are transmitted by the network. When the load reaches 80% of the bandwidth, the throughput of the lowest priority message begins to drop sharply. But when the offered load exceeds the bandwidth the throughput of the lowest priority message did not drop to zero as the formula in chapter four expressed. This is because at the beginning of the simulation the system is in an unstable state. Message queue is not built up. and lower priority messages still have some chance to be transmitted. FIgures 14 through 17 show the relationship between delay and priorities or delay and offered load. To show the relationship between the delay and priorities two or three loads are chosen to draw the curves. To show the relationship between the delay and load the highest and lowest priority messages in the given set of messages are chosen to draw the curves.

Figure 14 shows the maximum delay vs. message priorities and figure 15 shows the maximum delay vs. offered load. The trend is similar to the trend shown in figure 6 and figure 7 but the quantities are very different. The reason is that for the analytical result the maximum delay of each message

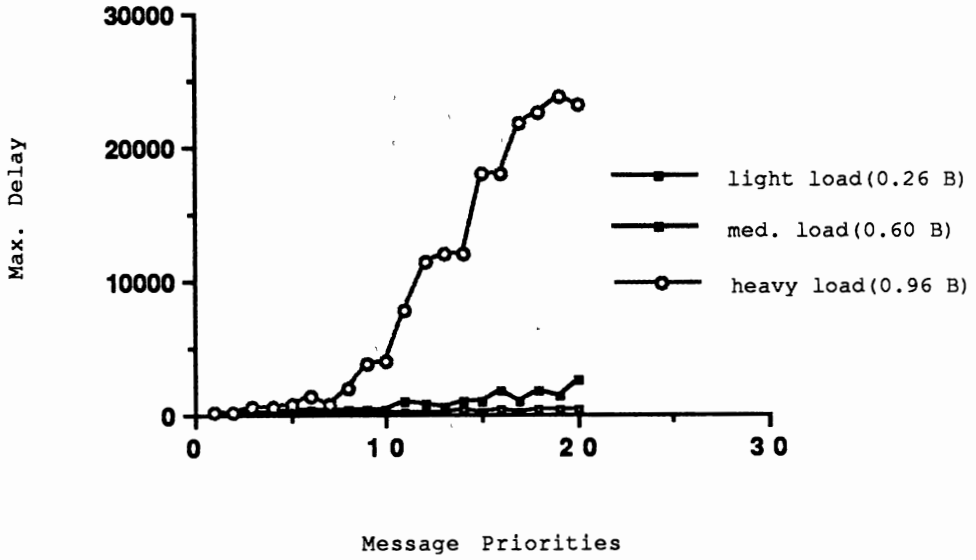


Figure 14. Maximum Delay vs. Message Priorities

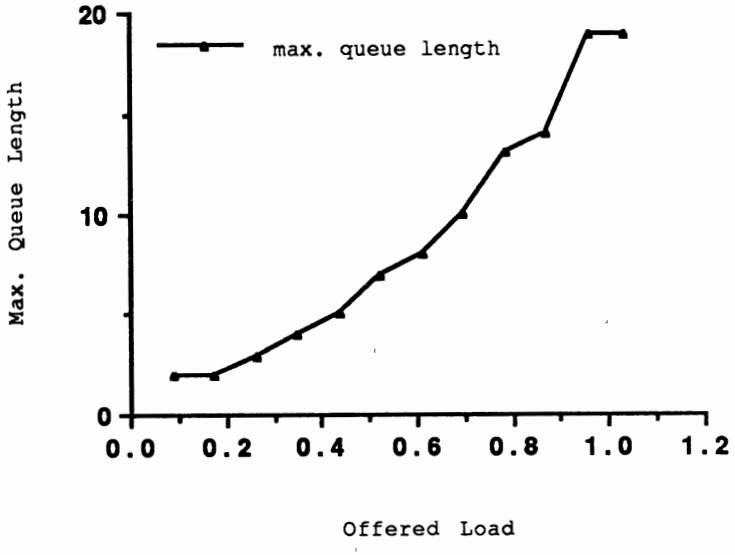


Figure 15. Maximum Delay vs. Offered Load

is obtained using the formula derived from the worst case situation, in which a message arrives at the communication channel with all higher priority messages at the same time and one of the longest messages just seized the communication channel. Besides, higher priority messages continue to arrive in their maximum arrival rate during the waiting time. The maximum delay of each type of message is deterministic. In the simulation program, the message arrivals are assumed to follow Poisson distribution. The program uses the function

$$F(x) = - (\frac{B}{\lambda} * \ln x)$$

to generate exponential message inter-arrival time. Here x is the uniformly distributed random number between 0 and 1 generated by the C library function. In order to simulate the worst case situation the random number generator must generate several 1s continuously to get several messages generated almost at the same time. This is impossible for the random number generator. It is known that the random number generator does not generate truly random numbers; instead it uses some algorithm to generate pseudo-random numbers. These algorithms insure that the same numbers will not be generated repeatedly. In this sense it can be said that the program is unable to simulate the worst case situation.

Figures 16 and 17 compare the simulation result with the result obtained by using the formulae in chapter four. Figure 16 shows the relationship between the expected delay and message priorities and figure 17 shows the relationship

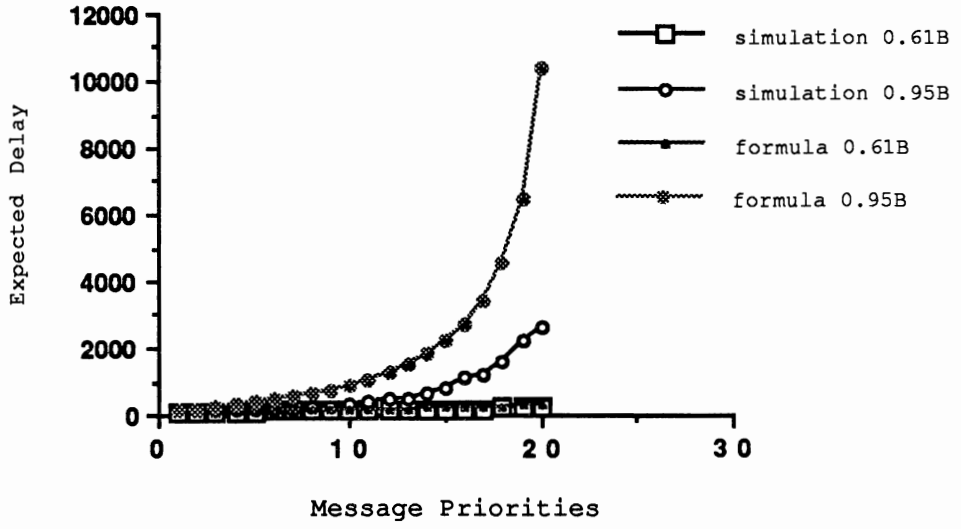


Figure 16. Expected Delay vs. Message Priorities

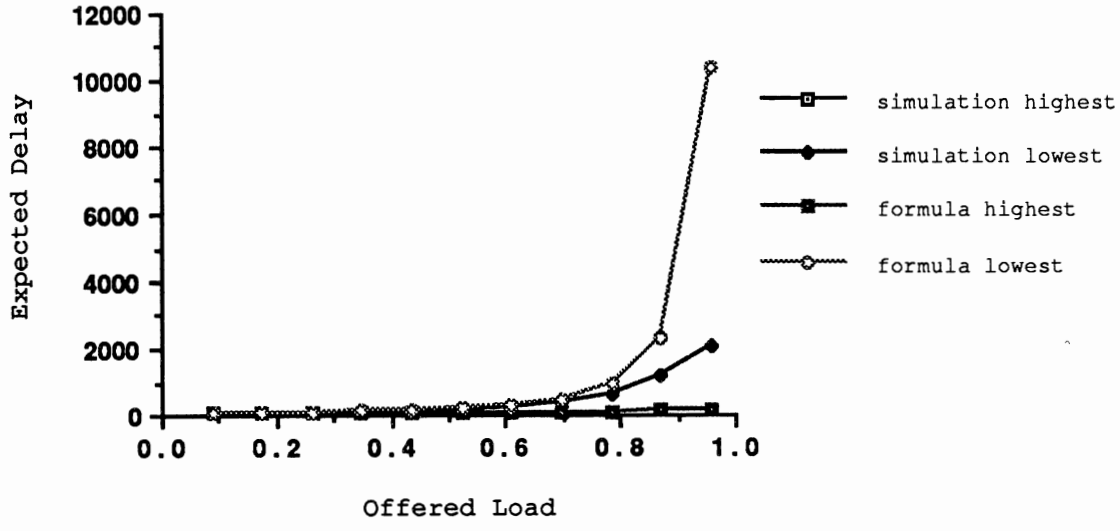


Figure 17. Expected Delay vs. Offered Load

between expected delay and the offered load. On figure 16 two different load are chosen to draw the curves. The words "simulation 0.61B" means the load is 61% of the bandwidth. The trend of the curves are the same but they do not completely agree. In figure 17 it can be seen that when the offered load reaches 70 percent of the bandwidth the two curves which represent the expected delay of the two lowest priority messages begin to separate. When the load reaches 100% of the bandwidth the difference increases to about five times. This is because on heavy load the lower priority messages have to wait a very long time before they get transmitted. While waiting the next message may be generated. The dual port RAM of the communication controller has only one communication object for each type of message. The newly generated message will either overwrite the old message or be discarded. The simulation program adopts the latter policy. In either situation the actual offered load drops and the traffic becomes less heavy. From figure 17 it can be seen that the delay of lower priority message is very sensitive to the load when the load exceeds 70% of the bandwidth. A small change in the load will cause a big change in the delay. Because the actual load drops, the expected delay drops by a relatively big amount. The difference between offered load and actual load is also shown in figure 12 where when the offered load exceeds the bandwidth the throughput is still less than the bandwidth. Another reason for this difference is that, the formulae are built according to queuing model. For the queuing model to

be correct, the traffic intensity ρ should be much smaller than 1. When the value of ρ is close to 1, the model is no longer accurate and the formulae are no longer realistic.

Figure 18 shows the maximum queue length vs. offered load. The relationship of the two is almost linear. Figure 19 shows the relationship between the number of collisions and offered load for the above particular data sets. Collisions have little influence on the throughput of the network because it wastes no bandwidth. The number of collisions depends not only on the offered load but depends on the distribution of messages. For example, there are three different priority messages waiting to be transmitted. If all of them are from one station then there will be no collision, because the station will send them one by one according to their priorities. If they come from three different stations then there will be three collisions. If they come from two different stations then there will be one or two collisions.

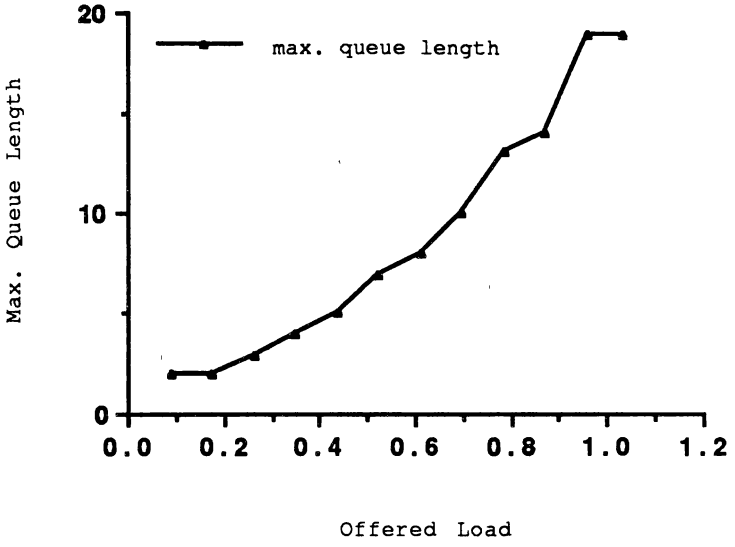


Figure 18. Max. Queue Length vs. Offered Load

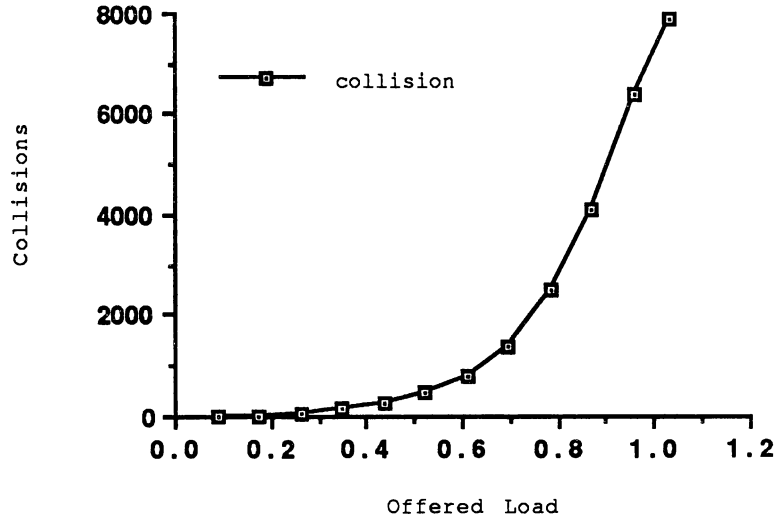


Figure 19. Number of Collisions vs. Offered Load in a CAN Network

CHAPTER VI

SUMMARY AND CONCLUSIONS

The Controller Area Network is a type of small local area network suitable for real-time distributed control systems. It has many special features. It is simple, inexpensive, deterministic, and easy to implement. It has short message latency for higher priority messages and flexible configuration and uses nondestructive message arbitration. The most popular local area networks like CSMA/CD type networks, token bus networks and token ring type networks are not suitable for real-time control systems. Compared with CAN these networks are too expensive, too complex, lack flexibility and have no adequate priority scheme to distinguish more important messages from less important ones. These networks are suitable for transferring big files, not suitable for transferring short and frequently transmitted control messages. When used in real-time control systems like in robots, cars, and airplanes, CAN is superior to these popular standard networks.

In this thesis the performance analysis of CAN has been made. Messages are assumed to be generated according to a Poisson distribution. Message transmission errors are ignored and the transmitted frames are assumed to be data frames and remote frames only. The analysis result may give the network designer a little more accurate description of

the behavior of the network and help him or her to select optimum design parameters.

Simulation of CAN has also been made to verify the analytical result. This program can also be used as a tool to help the CAN designer to evaluate his or her design.

Message priorities have great influence on the delay of message transmission when the network is heavily loaded. The priority assignment algorithm makes sure that there are solutions to given design requirements and helps to find the optimum priority assignment scheme.

For right now there are the performance models, the simulation program, and the message priority assignment algorithm to help the network designer to design the network. It would be better if there is one more tool which will allow the control system designer to monitor several control points simultaneously and to debug the control programs through the network. It is possible to attach a PC to the network and use Microsoft Windows software development kit or some other software to construct such programs and this would complete the basic design tool set for the CAN design.

BIBLIOGRAPHY

[Arne87] Arnett, J. D. , A High Performance Solution for In-Vehicle Networking-'Controller Area Network(CAN)', Earthmoving Industry Conference, Peoria, IL, April 7 1987. SAE paper #870823.

[Bick89] Bickerton,J. M. , & Chanham, R. , Simulation for Automotive System, International Congress and Exposition, Detroit, Michigan, Feb. 25 - March 3 1989.

[Deit90] Deitel, Harvey M. , An Introduction to Operating Systems, 2nd Edition, Addison Wesley Publishing company, Massachusetts, p. 295.

[Eyho89] Ey, Horst, Controller Area Network(CAN) Components, Electronic Component & Applications, vol. 9, no. 3, 1989 pp. 155-158.

[Gupt88] Gupta, Asnjay, CAN Facilitates In-Vehicle Serial Communications, On Board Communications for Machinery & Control, American Society of Agricultural Engineers paper #88-1649.

[Haye84] Hayes, j. F. , Modeling and Analysis of Computer Communication Networks, Plemum Press, New York, 1984.

[Hamm86] Hammond, J. L. , & O'Reilly, P. J. P. , Performance analysis of Local Computer Networks, Addision-Wesley Publishing Company, Reading Massachusetts, 1986.

[IEEE85a] IEEE Standard 802. 3 - 1985. Carrier Sense Multiple Access with Collision Detection, CSMA/CD 1985.

[IEEE85b] IEEE Standard 803. 4 - 1985. Token-Passing Bus Access Method and Physical Layer Specification, 1985.

[IEEE85c] IEEE Standard 802. 5 - 1985. Token-Passing Ring Access Method and Physical layer Specification, 1985.

[Inte88] Inter Corp. Auto-Communication Chip Replace Bulky Wires, Design News vol. 44, Aug. 88, p120.

[Iver88] Iversen, Wesley R. , Inter Gets a jump on the Auto Multiplex Market, Electronics, vol. 61, March 88, pp. 31-32.

[Jord88] Jordan, Pat, Controller Area Network, Electronics & Wireless World, vol. 94, Aug. 88, pp. 816-819.

[Jurg86] Jurgen Ronald K. , Coming from Detroit: Network on Wheels, IEEE Spectrum June 1986, pp. 53-59.

[Kien86] Kiencke, U. , Dais, S. , & Litschel, M. , Automotive Serial Controller Area Network, International Congress and Exposition, Detroit, Michigan, Feb. 24-28, 1986. SAE paper #860391.

[Klei75] Kleinrock, L. , Queuing Systems Volume I: Theory, John Wiley & Sun, Inc. , New York, 1975.

[Manu88] CAN 8025 Demo Board Manual, March 15, 1988.

[Mcmi75] McMillan, C. & Gonzalez, R. F. , System Analysis: A Computer Approach to Decision Models, 3rd Edition, Richard D. Irwin, Inc. ,Homewood, Illinois, 1973.

[Phai88] Phail, F. H. , Controller Area Network - An In-Vehicle Network Solution, 1988 International Summer Meeting of the American Society of Agricultural Engineers, Paper #88-3021.

[Stal90] Stallings, W. , Local Network, 3rd Edition, Macmillan Publishing Company, New York, 1990.

[Tane89] Tanenbaum A. S. , Computer Network, 2nd Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

APPENDIXES

APPENDIX A

NOTATIONS

λ : Average message generate rate of the whole system.

λ_i : Average message generate rate of i-th priority message.

μ : Service rate, message/sec = B/l

ρ : Traffic intensity $\rho = \lambda/\mu$.

B : Bandwidth of the network.

D : Total delay of all message in the network.

D_i : Maximum delay of i-th priority message.

d_i : Expected delay of i-th priority message.

AD_i : Allowed maximum delay of i-th priority message.

AVD : Average maximum delay of all messages.

F : expected frame length of the system.

F_i : Frame length of i-th priority message.

F_{max} : Maximum frame length in the system which is 108 bits.

G : Offered load of the network.

G_i : Offered load of i-th priority message.

IG : Input load of the network.

IG_i : Input load of i-th priority message.

IFS : Minimum number of interframe spaces between two frames which is 3 bits. .

L : Expected length of message in the system.

L_i : Length of i-th priority message.

L_{max} : Maximum message or data field length which is 64 bits.

M_i : Maximum message generate rate of i-th priority message.

n_s : Expected number of message in the system.

n_b : Expected number of message on the bus.

OHF : Overhead bits in each frame which is 47 bits

P(n) : Probability of n messages in the system.

Q : Maximum length of the queue.

q : Expected number of messages in the queue.

qt : Expected time of messages in the queue.

S : Throughput of the network.

S_i : Throughput of i-th priority message.

ES : Effective throughput of the network.

ESi : Effective throughput of i-th priority.

ti : Transmission time of i-th priority message. It is the time need
to transmit that message.

U : Channel utilization of the network.

W_i : Maximum waiting time of i-th priority message.

w_i : Expected waiting time of i-th priority message.

APPENDIX B

SIMULATION PROGRAM

```
/* Global.c */
/* global variable definition */


/* some flag value */
#define YES         1
#define NO          0
#define BUSY        1
#define IDLE        0

/* constant value */
#define FM_L       13     /* maximum fram length */
#define OBJ_L      11     /* maximum object length, include id
                             2 bytes */
                          /* control 1 byte, data segment at
                             most 8 bytes */
#define IDCON       3     /* length of message ID and control
                             field */
#define M_MSGS     10     /* maximum number of message a node
                             can */
                          /* transmit or receive */


/* segment flag value */
#define NUL         0     /* not transfer or receive, point to
                             no segment */
#define STR         1     /* for star bit */
#define ID          2     /* message ID field */
#define CON         3     /* control field */
#define DAT         4     /* data field */
#define CRC         5     /* CRC field */
#define ACK         6     /* acknoledge field */
#define END         7     /* end of fram field */
#define SPA         8     /* inter fram space */


/* network node */
typedef struct  markadd{
char  tra_obj[M_MSGS][OBJ_L],
                          /* store at most 10 objects for
                             transmiting */
                          /* for each object : ID-2 bytes,
                             control field- */
                          /* 1 byte, data-at most 8 bytes */
      tra_fram[FM_L];     /* before trasfered the star bit,
                             CRC field, */
                          /* acknowledge field, end mark are
                             added to */
                          /* the message thus makes a complete
                             fram then */
                          /* put into this buffer for
                             transmiting */

                          /* a node can transfer at most 10
```

```
                                types of msg. */
int     tra_id[M_MSGS],     /* 10 message IDs */
        tra_num[M_MSGS],    /* average number will be transfered
                               per second. */
        tra_len[M_MSGS],    /* length of each message in byte*/
        tra_don[M_MSGS],    /* total number already done now */
        tra_mdl[M_MSGS],    /* maximum delay recored */
        tra_yes[M_MSGS];    /* tra_yes[i] : 0-no ith type of
                               message  */
                            /* to transfer, 1-have message to
                               transfer */
long float
        tra_cdl[M_MSGS],    /* cumulated tra. delay. for compute
                               average */
        gen_tim[M_MSGS];    /* message generation time moment */

char    rec_buf[FM_L];      /* buffer for receiving messages.
                               after a */
                            /* message be received the receive
                               count */
                            /* will increment and the message is
                               thought */
                            /* to be consumed */
int     rec_id[M_MSGS],     /* 10 IDs for messages this node
                               will receive */
        rec_don[M_MSGS];    /* total number received */


char    prv_bus,            /* private bus. logical prodct of
                               all go to bus */
        prv_bus_flag;       /* private bus flag. 0-idle,
                               1-busy */
                            /* logical sum is the system
                               bus_flag */

int     tra_types,          /* how many types of message will be
                               transfered */
        tra_want,           /* this node has message to transfer
                               */
        transfer,           /* this node is transfering */
        tra_seg,            /* which segment a node is
                               transfering */
        tra_seg_bit,        /* which bit of a segment a node is
                               transfering */
        tra_dt_bits,        /* data field length in bit of
                               current message */
        tra_cur_id,         /* id of current transfering message
                               */

        rec_types,          /* how many types of message need to
                               be received */
/*      rec_want,  */       /* this node want receive message */
        receive,            /* this node is receiving message */
        rec_seg,            /* which segment a node is
```

```
                              transfering */
      rec_seg_bit,        /* which bit of a segment a node is
                              transfering */
      rec_dt_bits;        /* data field length in bit of
                              current message */


/* statistics variables */
int   tra_count,          /* total message transfered */
      rec_count;          /* total message received */
      } node_type ;


/* the following is the globle variables for the CAN */
/* simulation program. */


/* network architecture and parameters */
long float
      bandwidth,          /* bus bandwidth */
      finish,             /* run time expressed as bit time */
      s_clock,            /* system clock start from 0 to
                              finish */
      queue_l,            /* length of the queue, include
                              transmiting msg. */
      ave_arrival;        /* average number of messages arrive
                              to */
                          /* to the bus per second */

char  bus,                /* the system bus */
      bus_flag;           /* system bus flag */

int   total_nodes,        /* total nodes on the network */
      run_time;           /* run time of a simulation in
                              second */


/* statistics variables and accounters */
unsigned long int
      m_queue,            /* maximum queue length */
      tra_count,          /* total finished msg. transmission
                              of all nodes */
      rec_count,          /* total messages received by all
                              nodes */
      str_count,          /* total msg. transfer started */
      gene_count,         /* total message generated */
      coll_count;         /* collision counter */

long float
      idle_count,         /* system bus idle time in bit time
                              */
      busy_count;         /* system bus busy time in bit time
                              */
```

```
/* globle file pointer */
FILE        *ipr,       /* input file pointer */
            *opr;       /* output file pointer */



/* Program Main.c */
#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <alloc.h>      /* for dynamic memory allocation */
#include "global.c"     /* globle variable and data type */
#include "get_parm.c"
#include "sys_init.c"
#include "what.c"
#include "node_tra.c"
#include "node_rec.c"
#include "msg_gene.c"
#include "bit_tra.c"
#include "bit_rec.c"
#include "msg_cycl.c"
#include "final_ou.c"
#include "check_pm.c"


void main(void)
/* This program simulate Control Area Network */

  {
  node_type     *head ;
  long int          i,  j  ;
  char          infile[20] ,    /* input file */
         outfile[20] ;   /* output file name */

  /* clearn screen */
  for(i=0; i<24; i++) printf("\n") ;

  /* input network parameters */
  printf("\n\%s\n\n%s\n%s\n",
  "                    CONTROL AREA NETWORK SIMULATOR",
  "            This program simulate the Control Area
Network",
  "            Please specify the network you want to
simulate.");


  /* get input and output file names */
  printf("\nPlease give the input file name  : ") ;
  scanf("%s", infile) ;
  printf("\nPlease give the output file name : ") ;
```

```
  scanf("%s", outfile) ;

  /* open the input and output file */
  if ( (ipr=fopen(infile, "r") ) == NULL)
    {
    printf("\nInput file open fail") ;
    return ;
    }

  if ( (opr=fopen(outfile, "w")) == NULL )
    {
    printf("\nOutput file open fail") ;
    return ;
    }


  fprintf(opr,"\n%s", "How many second do you want to
simulste the network : ") ;
  fscanf(ipr, "%d", &run_time ) ;
  fprintf(opr, "  %d",  run_time ) ;

  fprintf(opr,"\n%s", "       How many bit per second is the
bus bandwidth : ") ;
  fscanf(ipr, "%d", &i ) ;
  bandwidth = i ;
  fprintf(opr, "  %f",  bandwidth ) ;

  fprintf(opr,"\n%s", "             The number of nodes on
the network are : ") ;
  fscanf(ipr, "%d", &total_nodes ) ;
  fprintf(opr, "  %d",  total_nodes ) ;

  head = (node_type *) calloc(total_nodes,
sizeof(node_type)) ;

  get_parm(total_nodes, head);

  /* check if there are logical error on the parameters */
  if (check_pm(total_nodes, head) == 0)
    {
    free(head) ;
    fclose(ipr) ;
    fclose(opr) ;
    return ;
    }

  /* initiate system variables */
  sys_init(total_nodes, head) ;

  /* run the network till the s_clock expired */
  while(s_clock < finish )
      msg_cycl(head) ;

  /* output final result */
```

```
  final_out(head) ;

  free(head) ;

  fclose(ipr) ;
  fclose(opr) ;

  printf("\n\n                SIMULATION FINISHED ") ;

  return ;
  }



void sys_init(int total_nodes, node_type  *head)
/* this program initiate globle variables and node variables
*/
  {
  int        i,  j,  k,  temp ;
  node_type  *p ;

  /* statistics variable */
  m_queue    = 0 ;
  tra_count  = 0 ;
  rec_count  = 0 ;
  str_count  = 0 ;
  gene_count = 0 ;
  idle_count = 0 ;
  busy_count = 0 ;
  coll_count = 0 ;

  /* initiate globle variables */
  s_clock = 0 ;
  finish = bandwidth * run_time ;
  bus = 1 ;
  bus_flag = IDLE ;
  queue_l = 0 ;

  /* calculate average number of messages arrival to the */
  /* bus every second */
  p = head ;
  ave_arrival = 0 ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<M_MSGS; j++)
      ave_arrival += p->tra_num[j] ;


  /* initiate network nodes */
  p = head ;


  for(i=0; i<total_nodes; i++, p++)
    {
```

```
    /* for every node sort transfering message according
       their IDs */
    for(j=0; j<p->tra_types-1; j++)
      for(k=0; k<p->tra_types-j-1; k++)
     if (p->tra_id[k] > p->tra_id[k+1])
       {
       temp = p->tra_id[k] ;
       p->tra_id[k] = p->tra_id[k+1] ;
       p->tra_id[k+1] = temp ;

       temp = p->tra_num[k] ;
       p->tra_num[k] = p->tra_num[k+1] ;
       p->tra_num[k+1] = temp ;

       temp = p->tra_len[k] ;
       p->tra_len[k] = p->tra_len[k+1] ;
       p->tra_len[k+1] = temp ;
       }

    /* form communication object */
    for(j=0; j<p->tra_types; j++)
      {
      /* message ID */
      /* 11 bits ID followed by 1 bit RTR, value 1, total 12
         bits */
      /* stored in the first 2 bytes, right aligned */
      p->tra_obj[j][0] = ( p->tra_id[j] & 0x780 ) >> 7 ;
      p->tra_obj[j][1] = ( (p->tra_id[j] & 0x7f) << 1 ) |
0x1 ;

      /* contrlo field, total 6 bits. 2 reserve bits, value
0, followed */
      /* by 4 bits to represent data fied length in byte. */
      p->tra_obj[j][2] = p->tra_len[j] & 0xf   ;

      /* put ID in the data field */
      p->tra_obj[j][3] = 'I' ;
      p->tra_obj[j][4] = 'D' ;
      p->tra_obj[j][5] = ' ' ;
      p->tra_obj[j][6] = ':' ;
      p->tra_obj[j][7] = ' ' ;
      p->tra_obj[j][8] = (p->tra_id[j] & 0xff00) >> 8 ;
      p->tra_obj[j][9] = p->tra_id[j] & 0xff ;
      p->tra_obj[j][10] = '\0' ;
      }
      /* end of communication object */


    /* other node variables */
    for(j=0; j<M_MSGS; j++)
      {
      p->tra_don[j] = 0 ;
      p->tra_mdl[j] = 0 ;
      p->tra_yes[j] = 0 ;
```

```
      p->tra_cdl[j] = 0 ;
      p->gen_tim[j] = 0 ;

      p->rec_don[j] = 0 ;
      }

    p->prv_bus = 1 ;
    p->prv_bus_flag = IDLE ;

    p->tra_want = NO ;
    p->transfer = NO ;
    p->tra_seg  = NUL ;
    p->tra_seg_bit = 0 ;

    p->receive  = NO ;
    p->rec_seg  = NUL ;
    p->rec_seg_bit = 0 ;

    p->tra_dt_bits = 0 ;
    p->rec_dt_bits = 0 ;

    /* node statistics */
    p->tra_count = 0 ;
    p->rec_count = 0 ;

    } /* end of one node */

  }



void  get_parm(int total_nodes, node_type  *head)
/* this program accept parameters about node message */
/* total_nodes : number of nodes on the bus */
/* head : pointer pointing to  node  array */

  {
  int          i,   j,  total_type ;
  node_type  *p ;

  /* clear arrays for accept parameters */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<M_MSGS; j++)
      {
      p->tra_id[j] = 2047 ;
      p->tra_num[j] = 0 ;
      p->tra_len[j] = 0 ;

      p->rec_id[j] = 0 ;
      }
```

```
  /* for every node */
  /* the message to be transfered and to be received */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    {
    fprintf(opr,"\n\n\n\n\n") ;
    fprintf(opr,"\n                          ======== Network
Node %d ======== ", i );

    fprintf(opr,"\n\n%s",
    "                          ---- Transfering Messages ----
") ;

    fprintf(opr,"\n\nTypes of message this node will
transfer : ");
      fscanf(ipr,"%d", &total_type ) ;
    fprintf(opr, "  %d ", total_type ) ;
    p->tra_types = total_type ;


    /* for every message to be transfered */
    /* their id, average number per second, length */
    for(j= 0; j<total_type; j++)
      {
      fprintf(opr,"\nnode %d transfering message number %d -
--- ", i, j) ;
      fprintf(opr,"\n                          message ID : ") ;
          fscanf(ipr,"%d", &p->tra_id[j]) ;
      fprintf(opr, "  %d", p->tra_id[j] ) ;

      fprintf(opr,"\naverage # of message per second : ") ;
          fscanf(ipr,"%d", &p->tra_num[j]) ;
      fprintf(opr, "  %d", p->tra_num[j] ) ;

      fprintf(opr,"\n          length of this message : ") ;
          fscanf(ipr,"%d", &p->tra_len[j] ) ;
          fprintf(opr, "  %d",  p->tra_len[j] ) ;

      }

    fprintf(opr,"\n\n%s",
    "                          ---- Receiving Messages ----
") ;

    /* for every message to be received by this node */
    /* message id, average number to be received */
    fprintf(opr,"\n\nTypes of message this node will receive
: ");
      fscanf(ipr,"%d", &total_type) ;
    fprintf(opr, "  %d", total_type ) ;
    p->rec_types = total_type ;

    for(j= 0; j< total_type; j++)
      {
```

```
      fprintf(opr,"\nnode %d receiving message number %d ---
- ", i, j) ;
      fprintf(opr,"\n                          message ID : ") ;
          fscanf(ipr,"%d", &p->rec_id[j]);
          fprintf(opr, "  %d", p->rec_id[j] ) ;
      }
    }  /* end for all nodes */

    return ;

  }



int check_pm(int total_nodes, node_type *head)
/* this function check if there are errors in input
parameters */
/* and stop the program or give warnings to the user */

  {
  node_type   *p ;
  int         tt_send,  i, j, k, l, give ;
  int         *send_id, *send_ck;

  /* find out how many types of transfer messages */
  p = head ;
  tt_send = 0 ;
  for(i=0; i<total_nodes; i++, p++)
    tt_send += p->tra_types ;

  /* allocate array to store message IDs */
  send_id = (int *) calloc(tt_send, sizeof(int)) ;
  send_ck = (int *) calloc(tt_send, sizeof(int)) ;

  /* initiate two arrays */
  for(i=0; i<tt_send; i++)
    {
    send_id[i] = -1 ;
    send_ck[i] = 0  ;
    }

  /* check for duplicate message ID */
  l = 0 ;
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<p->tra_types; j++)
      {
      for(k=0; k<l; k++)
     if ( p->tra_id[j] == *(send_id + k) )
       {
       printf("\n\nFATAL INPUT ERROR : ") ;
       printf("\nTwo types of message have the same ID :
%d",p->tra_id[j] );
```

```
        printf("\n---- Program Terminated ----\n\n") ;
        free(send_id) ;
        free(send_ck) ;
        return(0) ;
        }

       send_id[l] = p->tra_id[j] ;
       l += 1 ;
       }

  /* check if a node receive messages send by itself */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<p->rec_types; j++)
      for(k=0; k<p->tra_types; k++)
     if (p->rec_id[j] == p->tra_id[k])
       {
       printf("\n\nINPUT ERROR : ") ;
       printf("\nNode %d receive msg. %d send by itself
",i,p->rec_id[j]);
       printf("\n---- Program Terminated ----\n\n") ;
       free(send_id) ;
       free(send_ck) ;
       return(0) ;
       }

  /* check if every message transfered has node to receive
it */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<p->rec_types; j++)
      for(k=0; k<tt_send; k++)
     if( p->rec_id[j] == send_id[k] )
       send_ck[k] += 1 ;

  for(i=0; i<tt_send; i++)
    if( send_ck[i] == 0 )
      printf("\nWANRING : no node receive message %d\n",
send_id[i] );

  /* check if every message to be received by some nodes has
node to */
  /* send it */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<p->rec_types; j++)
      {
      give = 0 ;
      for(k=0; k<tt_send; k++)
     if (p->rec_id[j] == send_id[k] )
       give = 1 ;

      if (give == 0 )
```

```
     printf("\nWARNING : no node send message %d for node %d
to receive\n",
          p->rec_id[j], i ) ;
      }

  free(send_id) ;
  free(send_ck) ;

  return(1) ;

  }



void msg_cycl(node_type *head)
/* this program simulate one message transmition */
/* system bus_flag become IDLE is the end of a message cycle
*/
/* p->transfer : YES a node is speaking, NO stop */
/* p->receive  : YES a node is listening, NO stop */
  {
  int           cyc_end, i, j ;
  node_type     *p ;
  static unsigned long int
          next_arrival = 0 ;   /* next message arrival time
*/


  /* set some initial condition for a message cycle */
  /* at begining of a message cycle every node listen till
*/
  /* finish ID, some node stop listening. every node having
messages */
  /* to transfer can speak till arbitrition loss */
  bus_flag = IDLE ;
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    {
    p->transfer = YES ;
    p->receive  = YES ;
    p->tra_seg  = NUL ;
    p->rec_seg  = NUL ;
    }

  /* check and mark which node want to transfer message */

  queue_1 = 0 ;

  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    {
    p->tra_want = NO ;
    for(j=0; j<M_MSGS; j++)
```

```
      if (p->tra_yes[j] == YES)
     {
     p->tra_want = YES ;
     queue_l += 1 ;
     }
    }


  /* record maximum queue length */
  if (queue_l > m_queue)
    m_queue = queue_l ;


  /* if a node has nothing to say right now then it cannot
speak */
  /* after the message cycle begin */
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    if (p->tra_want == NO)
      p->transfer = NO ;


  /* at this point all node begin to listen--receive==YES */
  /* all node having message to transfer begin to speak--
transfer==YES */
  cyc_end = NO ;
  while(cyc_end == NO)
    {
    s_clock += 1 ;
    if (s_clock >= finish )
      return ;

    /* if clock reach the next arrival time generate a
message */
    /* and get new arrival time */
    if (s_clock >= next_arrival)
      {
      p = head ;
      next_arrival = s_clock + msg_gene(p) ;
   /*
      printf("\nnext message arrival time : %d ",
next_arrival ) ;
    */
      }


    /* transfer a bit to the bus */
    p = head ;
    bit_tra(p) ;

    /* receive a bit from bus */
    if (bus_flag == BUSY)
      {
      p = head ;
```

```
      bit_rec(head) ;
      cyc_end = NO ;

      /* statistics */
      busy_count += 1 ;
      }
    else
      /* the system bus_flag is IDLE no message on bus */
      {
      cyc_end = YES ;

      /* statistics */
      idle_count += 1 ;
      }
    }
  }



int msg_gene(node_type *head)
/* this function generate message for nodes to transfer */
/* it returns time interval for next message arrival if a
message */
/* is generated; 0 if no message generated */
  {
  int          i, k, interval ;
  node_type    *p ;
  time_t       t ;
  double       u_random, rate, second ;
  static int   node_no=0, msg_no=0 ;


  /* randomly choose a node */
  /* reduce the probability of choosing the same node twise
*/
  k = rand() % total_nodes ;
  if (k == node_no)
    k = rand() % total_nodes ;
  node_no = k ;
  p = head ;
  for(i=0; i<k; i++)
    p++ ;


  /* if this node does not transfer any message then return
*/
  if (p->tra_types == 0)
    return(0) ;

  /* randomly choose a type of message */
  /* try to find one which is not already waiting to
transfer */
  msg_no = 0 ;
```

```
  for(i=0; i< p->tra_types; i++)
    {
    k = rand() % p->tra_types ;
    if(p->tra_yes[k] == NO)
      msg_no = k ;
    }
    k = msg_no ;
    msg_no = p->tra_id[k] ;

  /* if did not find message which is not waiting to be */
  /* transfered then return  */
  if (p->tra_yes[k] == YES)
    return(0) ;

  /* if this type of message has transfered more than 110%
of */
  /* its average number per second then not generate more */
  /* p->tra_num[k]--average per second */
  /* p->tra_don[k]--number of alredy transferd kth type of
message */
  /* in the begining run time is 0, add 1.1 to avoid
denominate 0. */
  second = (s_clock + 1.1) / (bandwidth * 1.0) ;
  rate = p->tra_don[k] / second ;
  if (p->tra_num[k] * 1.1 >= rate)
    {
    p->tra_yes[k] = YES ;
    p->gen_tim[k] = s_clock ;

    /* generate interval for next message arrival */
    /* u_random is the uniform distributed random number
between 0 & 1 */
    /* interval is in bit time */
    u_random = rand() % 1000 ;
    if (u_random < 5.0)
      u_random = 5.0 ;
    u_random = u_random/1000 ;
    interval = -(log(u_random)* bandwidth/ave_arrival) ;

      printf("\nnode %3d message %4d geneted at time %f ",
            node_no,  msg_no,    s_clock);

    /* statistics */
    gene_count += 1 ;

    return(interval) ;
    }
  else
    return(0) ;

  }
```

```
void bit_tra(node_type *head)
/* transfer a bit to the bus. this will involve all nodes
which */
/* wants to transfer. 0 will overwrite 1. the result is
correct */
/* system bus content and bus_flag content. */
  {
  int          i ;
  node_type  *p ;

  /* initiate private bus and its flag */
  p = head ;
  for(i=0; i < total_nodes; i++, p++)
    {
    p->prv_bus = 1 ;
    p->prv_bus_flag = IDLE ;
    }

  /* all node transfer */
  p = head ;
  for(i=0;  i<total_nodes; i++, p++)
    if (p->transfer ==  YES)
      node_tra(i, p) ;

  /* get transmit result by collect each node's result */
  /* on bus 0 is the dominent bit, 0 overwrite 1  */
  /* on bus_flag 1 is dominent bit, 1 overwrite 0 */
  bus = 1 ;
  bus_flag = IDLE ;
  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    {
    bus = bus & p->prv_bus ;
    bus_flag = bus_flag | p->prv_bus_flag ;
    }
  }



void node_tra(int node_no, node_type *p)
/* transfer a bit to private bus */
  {
  int    i,
      j,        /* pointer to which message this node is
transfering */
      k ,       /* current byte */
         delay ;   /* message transfer delay */

  /* at begining of a message bus is idle : bus_flag == IDLE
*/
  if (bus_flag == IDLE)
    {
```

```
    /* find the highest priority message which need to be
transfered */
    for(i=M_MSGS-1; i>=0; i--)
      if ( p->tra_yes[i] == YES )
     j = i ;

    /* record current message id */
    p->tra_cur_id = p->tra_id[j] ;


      printf("\nnode %3d message %4d started at time %f",
          node_no, p->tra_id[j], s_clock ) ;
    /*
    for(i=0; i<OBJ_L; i++)
      printf("%3d-", p->tra_obj[j][i] ) ;
    */


    /* assemble the fram in the fram buffer tra_fram */
    /* put ID, control file and data segment to tra_fram */
    k = p->tra_len[j] + IDCON ;
    for(i=0; i<k; i++)
      p->tra_fram[i] = p->tra_obj[j][i] ;

    /* align effective bits to left end of a byte */
    p->tra_fram[0] = p->tra_fram[0] << 4 ;
    p->tra_fram[2] = p->tra_fram[2] << 2 ;


    /* add CRC, 2 byte, to tra_fram */
    p->tra_fram[k] = 0xff ;
    k += 1 ;
    p->tra_fram[k] = 0xff ;

    /* get data field length in bit */
    p->tra_dt_bits = p->tra_len[j] * 8 ;

    /* set flags, pointer, counter */
    p->transfer = YES  ;
    p->tra_seg    = STR  ;
    p->tra_seg_bit  = 1    ;


    /* transfer star bit */
    p->prv_bus = 0 ;
    p->prv_bus_flag = BUSY ;

    /* statistics */
    str_count += 1 ;

    return ;
    }
```

```
  /* this node already begin to transfer message, so just
continue */
  /* when transfering a bit always take the left most bit
then shift */
  /* lower bit left */
  /* tra_seg and tra_seg_bit indicaes where the current bit
comes from */

  p->prv_bus_flag = BUSY ;
  switch (p->tra_seg)
    {
    /* start bit */
    case
    STR : p->prv_bus = p->tra_fram[0] & 0x80 ;
       p->prv_bus = p->prv_bus >> 7 ;
       p->tra_fram[0] *= 2 ;
       p->tra_seg = ID ;
       p->tra_seg_bit = 1 ;
       break ;
    case
    ID  : /* ID field has finished. next is the control
field */
       if (p->tra_seg_bit == 12)
         {
         p->tra_seg = CON ;
         p->tra_seg_bit = 0 ;
         k = 2 ;
         }
       else if (p->tra_seg_bit >= 4)
         k = 1 ;
       else
         k = 0 ;

       p->prv_bus = p->tra_fram[k] & 0x80 ;
       p->prv_bus = p->prv_bus >> 7 ;
       p->tra_fram[k] = p->tra_fram[k] << 1 ;
       p->tra_seg_bit += 1 ;
       break ;
    case
    CON : /* control field is 6 bits long. see if it is
finished */
       if (p->tra_seg_bit == 6)
         {
         p->tra_seg = DAT ;
         p->tra_seg_bit = 0 ;
         k = 3 ;
         }
       else
         k = 2 ;

       p->prv_bus = p->tra_fram[k] & 0x80 ;
       p->prv_bus = p->prv_bus >> 7 ;
       p->tra_fram[k] *= 2 ;
       p->tra_seg_bit += 1 ;
```

```
      break ;
    case
    DAT : /* length of data field in bit is stored in
tra_dt_bits */
      if (p->tra_seg_bit == p->tra_dt_bits)
        {
        p->tra_seg = CRC ;
        p->tra_seg_bit = 0 ;
        k = p->tra_dt_bits/8 + IDCON ;
        }
      else
        k = p->tra_seg_bit/8 + IDCON ;

      p->prv_bus = p->tra_fram[k] & 0x80 ;
      p->prv_bus = p->prv_bus >> 7 ;
      p->tra_fram[k] *= 2 ;
      p->tra_seg_bit += 1 ;
      break ;
    case
    CRC : /* CRC field is 16 bits long */
      if (p->tra_seg_bit == 16)
        {
        p->tra_seg = ACK ;
        p->tra_seg_bit = 1 ;
        p->prv_bus = 1 ;
        break ;
        }
      else
        k = IDCON + p->tra_dt_bits/8 + p->tra_seg_bit/8 ;

      p->prv_bus = p->tra_fram[k] & 0x80 ;
      p->prv_bus = p->prv_bus >> 7 ;
      p->tra_fram[k] *= 2 ;
      p->tra_seg_bit += 1 ;
      break ;
    case
    ACK : /* ack field consist of two 1s */
      if (p->tra_seg_bit == 2 )
        {
        p->tra_seg = END ;
        p->tra_seg_bit = 1 ;
        p->prv_bus = 1 ;
        break ;
        }
      p->prv_bus = BUSY ;
      p->tra_seg_bit += 1 ;
      break ;
    case
    END : /* end mark consists of 7 1s */
      if (p->tra_seg_bit == 7)
        {
        p->tra_seg = SPA ;
        p->tra_seg_bit = 1 ;
        p->prv_bus = 1 ;
```

```
          break ;
          }
        p->prv_bus = 1 ;
        p->tra_seg_bit += 1 ;
        break ;
    case
    SPA : /* inter-fram space is 3 '1's */
        if (p->tra_seg_bit < 3)
          {
          p->prv_bus = 1 ;
          p->tra_seg_bit += 1 ;
          }
        else
          {
          /* one message transmission is finished, change
status */
          p->prv_bus_flag = IDLE ;
          p->tra_want = NO ;
          p->tra_seg = NUL ;
          p->tra_seg_bit = 0 ;
          p->transfer = NO ;

          /* find message type */
          for(i=M_MSGS-1; i >=0; i--)
            if (p->tra_id[i] == p->tra_cur_id)
           j = i ;

          /* calculate delay & cumulated delay  */
          delay = s_clock - p->gen_tim[j] ;
          p->tra_cdl[j] += delay ;

          /* record max. delay */
          if (delay > p->tra_mdl[j])
            p->tra_mdl[j] = delay ;

          /* calcel tra. req. and msg. gen. time */
          p->tra_yes[j] = NO ;
          p->gen_tim[j] = 0  ;

          /* record one more msg. has transmited */
          p->tra_don[j] += 1 ;
          tra_count += 1 ;


          printf("\nmessage transfer ended") ;


          }
          /* end of if */
      }   /* end of switch */


  return ;
```

```
  }   /* end of function */



void bit_rec(node_type *head)
/* nodes which need to  receive message from the bus receive
*/
/* a bit from the bus */
  {
  int      i ;
  node_type  *p ;

  p = head ;
  for(i=0; i<total_nodes; i++, p++)
    if ( p->receive == YES )
      node_rec(i, p) ;

  return ;
  }



void node_rec(int node_no, node_type *p)
/* This function receive a bit form the  bus */
  {
  int   i,
     k ,              /* receiving buffer byte index */
     cur_id ;         /* current message ID */


  switch (p->rec_seg)
  /* before receive the current bit on  the bus */
  /* the rec_seg and rec_seg_bit point to the bit received
last time */
    {
    case
    NUL : /* begin to receive a new message */
       /* this bit is start bit */
       p->rec_seg = STR ;
       p->rec_seg_bit = 1 ;

       /* clearn the receive buffer */
       for(i=0; i<FM_L; i++)
         p->rec_buf[i] = 0 ;

       break ;
    case
    STR : /* the coming message bit is the first bit of ID
*/
       p->rec_seg = ID ;
       p->rec_seg_bit = 1 ;
       p->rec_buf[0] = bus ;
```

```
       /* message arbitrition */
       /* if this node is transfering message and what
received */
       /* is different from transfered then stop transfering
*/
       /* but still need to listen */
       if ( (p->transfer== YES) && (p->prv_bus != bus) )
         p->transfer = NO ;
       break ;
    case
    ID  : /* finish receiving ID field */
       if (p->rec_seg_bit == 12)
         {
         p->rec_seg = CON ;
         p->rec_seg_bit = 1 ;
         p->rec_buf[2] = bus ;
         break ;
         }

       /* message transmition arbitrition */
       /* if a node loss arbitrition it still need to listen
*/
       /* the current message may for him */
       if ( (p->transfer==YES) && (p->prv_bus != bus) )
         {
         p->transfer = NO ;
         printf("\nnode %3d loss arbitrition at time %f",
               node_no, s_clock ) ;

         /* statistics */
         coll_count += 1 ;
         }

       /* receive a message bit into the buffer */
       /* ID is in byte 0 and byte 1, decide which byte */
       if (p->rec_seg_bit >= 4)
         k = 1 ;
       else
         k = 0 ;
       /* first shift current byte left then put the coming
bit */
       p->rec_buf[k] = p->rec_buf[k] * 2 + bus ;
       p->rec_seg_bit += 1 ;

       /* first 11 bits of ID field constitute the message
ID */
       /* check if this node receive this message or */
       /* it is transfering */
       if ((p->rec_seg_bit == 11) && (p->transfer == NO))
         {
         p->receive = NO ;
         cur_id = p->rec_buf[0] * 128 + p->rec_buf[1] ;
         for(i=0; i<p->rec_types; i++)
```

```
            if (cur_id == p->rec_id[i])
             p->receive = YES ;
           }

        break ;
      case
      CON : /* control field 6 bits */
        if (p->rec_seg_bit == 6)
          {
          /* find out data field length in bit */
          p->rec_dt_bits = (p->rec_buf[2] & 0xf) * 8 ;

          /* data field may be 0 byte logn */
          if (p->rec_dt_bits > 0)
            p->rec_seg = DAT ;
          else
            p->rec_seg = CRC ;

          p->rec_seg_bit = 0 ;
          k = 3 ;
          }
        else
          k = 2 ;

        p->rec_buf[k] = p->rec_buf[k] * 2 + bus ;
        p->rec_seg_bit += 1 ;

        break ;
      case
      DAT : /* leng of data field in bit stored in
rec_dt_bits */
        if ( p->rec_seg_bit == p->rec_dt_bits )
          {
          p->rec_seg = CRC ;
          p->rec_seg_bit = 0 ;
          k = IDCON + p->rec_dt_bits/8 ;
          }
        else
          k = IDCON + p->rec_seg_bit/8 ;

        p->rec_buf[k] = p->rec_buf[k] * 2 + bus ;
        p->rec_seg_bit += 1 ;

        break ;
      case
      CRC : /* CRC field 16 bits */
        if ( p->rec_seg_bit == 16 )
          {
          p->rec_seg = ACK ;
          p->rec_seg_bit = 1 ;

          /* sent ACKNOWLEDGE back */
          bus = 0 ;
```

```
        break ;
        }

      k = IDCON + p->rec_dt_bits/8 + p->rec_seg_bit/8 ;
      p->rec_buf[k] = p->rec_buf[k] * 2 + bus ;
      p->rec_seg_bit += 1 ;

      break ;
    case
    ACK : /* acknowledge field 2 bits long */
      if ( p->rec_seg_bit == 2 )
        {
        p->rec_seg = END ;
        p->rec_seg_bit = 0 ;
        }

      p->rec_seg_bit += 1 ;

      break ;
    case
    END : /* end mark 7 '1's */
      if ( p->rec_seg_bit == 7 )
        {
        p->rec_seg = SPA ;
        p->rec_seg_bit = 0 ;
        }

      p->rec_seg_bit += 1 ;

      break ;
    case
    SPA : /* interfram space 3 bits, last bit will not
received */
      /* because in the 3th space bus_flag already set IDLE
*/
      if ( p->rec_seg_bit < 2 )
        p->rec_seg_bit += 1 ;
      else
      /* this is last bit of this cycle. a cycle is
finished */
      /* increment count, signal end of a message cycle */
        {
        cur_id = p->rec_buf[0] * 128 + ( p->rec_buf[1] >> 1
) ;
        for(i=0; i<p->rec_types; i++)
          if (cur_id == p->rec_id[i])
         p->rec_don[i] += 1  ;

        /* globle statistics */
        if (p->transfer == NO)
          rec_count += 1 ;

          printf("\nnode %3d message %4d received at time
%f",
```

```
                node_no, cur_id, s_clock ) ;

          return ;
          }

    }     /* end of switch */

  return ;

  }     /* end of the function */



void final_out(node_type *head)
/* output simulation result */
  {
  int      i,    j,
       iid,        /* message ID */
       ittra,      /* total transfered of one type */
       iadel,      /* average delay of  a type */
       imdel ;     /* maximum delay of a type */
  node_type   *p ;

  /* clear screen and output title */
  for(i=0; i<14; i++)
    fprintf(opr,"\n") ;

  fprintf(opr,"\n          OUTPUT OF THE SIMULATION
RESULT\n\n\n") ;

  /* general information */
  fprintf(opr,"\nTotal time in second  : %d", run_time) ;
  fprintf(opr,"\nTotal time in bit     : %f", finish  ) ;
  fprintf(opr,"\nBus busy time         : %f", busy_count );
  fprintf(opr,"\nBus idle time         : %f", idle_count ) ;
  fprintf(opr,"\nBus utilization       : %f",
busy_count*1.0/finish) ;
  fprintf(opr,"\nTotal collision       : %d", coll_count ) ;
  fprintf(opr,"\nTotal msg. generated  : %d", gene_count ) ;
  fprintf(opr,"\nTotal tra. started    : %d", str_count ) ;
  fprintf(opr,"\nTotal tra. completed  : %d", tra_count) ;
  fprintf(opr,"\nTotal msg. received   : %d", rec_count) ;
  fprintf(opr,"\nMaximum queue length  : %d", m_queue  ) ;


  /* output message information */
  p = head ;
  fprintf(opr,"\n\n%s",
  " Message ID     From Node   Total Transfered   Ave. Delay
Max. Delay");
  for(i=0; i<total_nodes; i++, p++)
    for(j=0; j<p->tra_types; j++)
```

```
    {
    iid   = p->tra_id[j] ;
    ittra = p->tra_don[j] ;
    iadel = p->tra_cdl[j] / ittra ;
    imdel = p->tra_mdl[j] ;
    fprintf(opr,"\n%10d%15d%15d%15d%15d", iid, i, ittra,
iadel, imdel) ;
    }

  return ;
  }
```

VITA

Zhengou Wang

Candidate for the Degree of

Master of Science

Thesis: ANALYSIS AND DESIGN OF CONTROLLER AREA NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Guiyang, Guizhou province, China, September 28, 1950, the son of Zhizan Wang and Daoxiang Xing.

Education: Graduated from Guiyang No 6 High School, Guiyang, Guizhou province, China, in August 1968; received Bachelor of Engineering Degree in Computer Science from Chongqing University in March, 1982; completed requirements for the Master of Science Degree at Oklahoma State University in December, 1991.

Professional Experience: Teaching Assistant, Department of Computer Science, Oklahoma State University, August, 1989 to May, 1991. Engineer, Computer Center of Tongji University, Shanghai, China, January, 1982 to August, 1988;