A TIME-DRIVEN DISCRETE-EVENT SIMULATOR FOR CONTROLLER AREA NETWORKS

By

MUJIBUR REHMAN MOHAMMAD

Bachelor of Engineering

R. V. College of Engineering

Bangalore, India

1991

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, 1994

A TIME-DRIVEN DISCRETE-EVENT SIMULATOR FOR CONTROLLER AREA NETWORKS

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

I thank my graduate adviser Dr. Mitchell L. Neilsen for the guidance, help, and the time he has given for the completion of my thesis work. His perseverance and hard work inspired me to venture into the advanced aspects of this work. Without the encouragement and help he has given me, the completion of this work would have been impossible. I also sincerely thank Dr. K.M.George and Dr. Lu for serving on my committee. Their suggestions have helped me to improve the quality of this work.

My respectful thanks goes to my parents Mr. M. A. Khaiyoom and Mrs. Ghousinnisa Begum for all the love and support they have given me in life. Also I would like to thank my elder sister Mrs. Fasiunnisa Begum for her support and encouragement by which I got inspiration to work hard and achieve this goal. And, last but certainly not least, I thank all other members of my family for the love, encouragement and confidence they have endorsed in me. Also I would like to thank my friends Shaik Jeelani, Tariq Irfan, Urfi Obaid, Fazlur Rahman, Humair Ahmed, Rehan Ahmed, Naeem Malik, Zeeshan Shafaq and Irfan Khalilullah for their constant encouragement in achieving this goal.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Day by day advancements in technology are making human life more and more comfortable. As a part of this revolution, computer networks are playing an important role in our daily lives. In this process, there has been an increase in the number of electronically controlled systems found in automobiles during the past several years. Also a trend towards down-sized, fuel efficient vehicles has placed a increasing emphasis on reducing the wiring harness caused by the growing number of electronically controlled systems in the cars [WJJ86]. Until recently, all electronic systems worked independently. Different control units measure and process sensor signals redundantly. In these situations, the distributed multivariable control approaches cannot be implemented [KL86]. We can optimize the combined system performance without introducing the additional hardware burden for sensors and actuators by constructing a high speed serial communication network which is an in-vehicle network. In this chapter we will discuss some existing in-vehicle networks.

1.1 C^2D Network

The C^2D Network was developed by Chrysler Corporation. Vehicle networks can be classified into three groups: Control Multiplexing, Data Communications and High Speed Controller. Control Multiplexing is a classical form. An example of Control Multiplexing is controlling the vehicle lamps. Control Multiplexing affects the base cost of the vehicle. Control Multiplexing will generally increase the cost and complexity while the safety and reliability is reduced [Mie86]. Data Communications interconnects the different modules such as engine controller, instrument cluster and other electronic modules. It doesn't affect the base wiring of the vehicle.

Vehicle multiplexing for intermodule data communications has a completely dif-

ferent effect on the complexity of the wiring harness. A single Intermodule Data Communications Network can be shared by all sensors. This reduces the harness size dramatically. For example, consider a vehicle distance sensor in a non-multiplexed vehicle. This sensor must be connected to the engine controller, trip computer and instrument cluster. This can be compared with the distance sensor in a data multiplexed vehicle network. In this case, the sensor is connected only to the engine controller and distance data is spread across the network along with the other sensor signals. By multiplexing data, there is a dramatic savings in the amount of wiring and complexity. Also multiplexed wiring in new models promise cost-effective control and accurate diagnostics [Jur86]. Data Multiplexing results in elimination of redundant hardware and firmware. The C^2D Network uses the Carrier Sense Multiple Access (CSMA) protocol. Below, Figure 1.1 shows the message format.

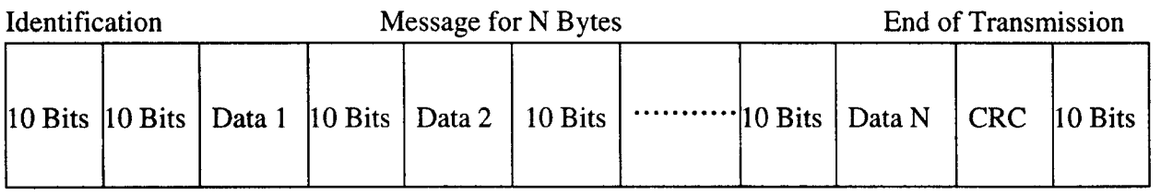

Figure 1.1 Message Format of C^2D Network [Mie86]

In the above message format, note that each data byte is separated by some idle bits. These idle periods are to allow the use of firmware control and direct connection to the host microcomputer's asynchronous serial I/O port. The identification byte of each message is unique. Message ID is transmitted first, followed by message data. The data is followed by a message CRC and is concluded by 10 bits which define the end of transmission. UARTNRZ was chosen as the technique used to encode bits [Mie86]. A baud rate of 7812.5 bits per second is chosen. This is because this speed of data permits a common clock for most microcomputers at their maximum frequency

of operation. Figure 1.2 below shows a block diagram of interfacing between the bus and the microcomputer.

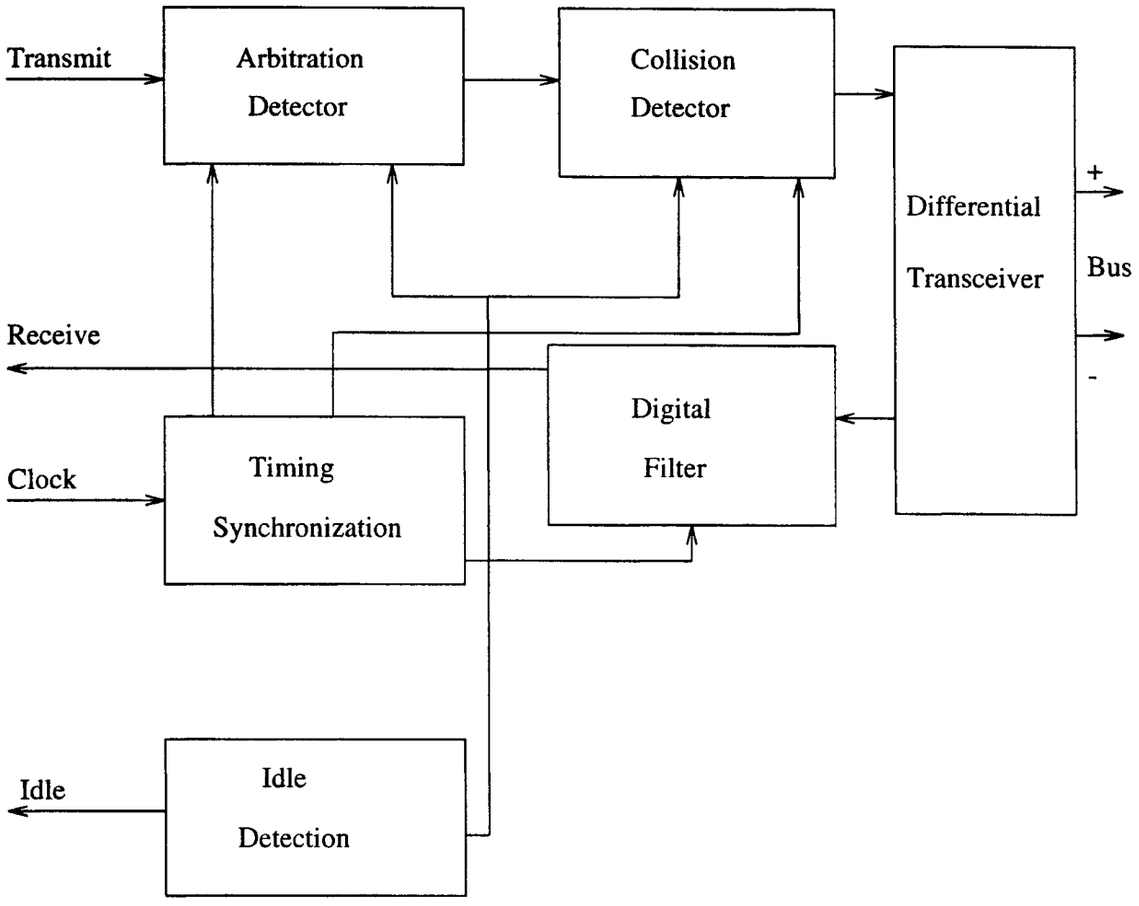

Figure 1.2 Block Diagram of C²D Bus Interface [Mie86]

In the above block diagram, the Arbitration Detector checks if arbitration is taking place between two messages; i.e., if two messages want to transmit at the same time. If so, the Arbitration Detector allows transmission on a first-come-first-serve basis. Where as the Collision Detector in the same above case, allows a message with highest priority to transmit. The Digital Filter will remove any electromagnetic interference (EMI) from the received data coming out of the bus. The Differential Transceiver is known as a contention permitting Differential Transceiver. This is a serial interface device which accepts digital signals and translates this information for transmission

on a two-wire differential bus. The Idle Detector detects whether the bus is idle or busy and feeds this information back to the microcomputer. This is accomplished by sensing a received stop bit followed by 10 continuous bits of idle or logic ones. The Timing Synchronization uses an external clock to establish the synchronization and baud rate the timing signal. This network has an advantage that the modules can be added or deleted without affecting the network. So this development in the Serial Communications Network can handle both Control Multiplexing and Data Communications.

1.2 Controller Area Network (CAN)

CAN is a real-time serial communications network. CAN is also a in-vehicle network which was developed to resolve the spacing problem raised by wiring, control units, sensors and other components. In 1985, Robert Bosch GmbH and Intel agreed to develop CAN. By 1987, the first sample of CAN was produced. CAN has the following properties whcih makes it suitable as a real-time network [Pha86].

- Prioritized bitwise arbitration for fast transmission of high priority messages with a latency time as short as 150 microseconds.
- High transmission rate in the range of 1 Mbps for a bus length of 40 meters.
- An open system which allows to add or delete any number of nodes without changes in the underlying software or hardware of any node.
- Data consistency.
- Multicast reception which allows the receipt of a message by any number of nodes.

- Multi-mastership that allows any node to start transmission when the bus is free.
- Automatic retransmission of corrupted messages as soon as the bus is idle again.
- Error detection and error signalling.
- Guarantee of latency times of messages.
- Separation in temporary errors and permanent failures of nodes and switching off of defective nodes.

Message transmission in CAN takes place as follows:

- First, a node checks to see if there is a message on the bus.
- If there is no message, it transmits its message immediately.
- If there is a message on the bus, then the message is delayed until the current message is completed.

The message identifier is the first component of a message sent by a node to the bus [WL92]. This contains both name and priority code for each message.

CAN is simple, easy to implement and cost effective. CAN reduces harness size and manufacturing complexity, eliminate sensors and facilitates in-vehicle electronic options. CAN can handle up to 2^{20} messages simultaneously with a maximum latency of 150 microseconds [Ey89]. It has high reliability, low latency, minimum CPU burden for communication, maximum transparency and data consistency. CAN is rapidly becoming an international standard. CAN is gaining momentum in Europe [Ive88]. Applications of CAN are mainly found in cars. Other applications of CAN include power train, factory automation, agricultural trucks, military and construction vehicles.

1.3 Simulation of CAN

A computer simulation can be defined as: Computer Simulation is the discipline of designing a model of an actual or theoretical physical system, executing on a digital computer, and analyzing the execution output [Fis92]. As simulation is a powerful tool to evaluate the performance of any network, CAN is also simulated using a computer program. Simulation aids in predicting changes in network performance and comparing alternate design. Simulation also provides information about various network parameters.

Simulation needs few assumptions and approximations of the network. To run simulation, a CAN must be described in terms of number of nodes, transmission speed, message identifiers, message length and noise. The objective of this research is to analyze the performance of CAN under different load constraints by using a Graphical User Interface (GUI) created for it. These performance characteristics include network load, network throughput, average response time, average turnaround time, number of missed deadlines, number of errors, number of retransmissions and bus utilization. Some other features of this simulation will include:

- Start times for each message type can be specified by the user. Start times may also be randomly distributed.
- Statistical summaries for each node in the network are produced.
- The user can enter a random number seed so that a particular simulation can be regenerated.
- The user can specify fixed time window for load calculations to obtain steady state behavior.
- The user can specify a particular input file for simulation.

- The user can specify the simulation time in bit time.
- The user can increase or decrease the simulation speed.
- The user can specify the transmission speed of the network.
- After the simulation is complete, the user can view the results in both texual and graphical form.

CHAPTER 2

PAST RESEARCH ON CAN SIMULATION

There are some CAN Simulation packages existing now. Philips has NetSim, a PC based simulation for which a CAN has to be described in terms of number of nodes, transmission speed, message identifiers, message length and some other factors. The output of this simulation provides network throughput, network delay and bus load. Also Robert Bosch GmbH has provided some on-board CAN simulator from which we can obtain the performance measures.

At Oklahoma State University, some research was done in the area of CAN simulation. CAN simulation research was perceived by Mr. Pennathur Nataraj on bitwise simulation of CAN. In this CAN simulation bit-by-bit logic is used. One of the key issue of this simulation is to map the physical time to the simulation time. A global clock forms the simulation time. Priorities are assigned to messages based on their schedule order. This research shows that under 30 percent of loads CAN works well [Nat93]. But on the whole the network performs well under 40 percent of loads. A throughput of 500 messages/sec is achieved for this load. It has been concluded in this research that there is a large idle time that can provide some additional loading time. Also stated in conclusion that response time is faster for loads lesser than 60 percent with average response times being less than 6 msec over a 100 msec run.

The other research done on CAN at Oklahoma State University is to analyze the CAN performance under asymmetric traffic loads. This research was perceived by Ms. Tsao-Jean Leu. In her research work the following assumptions were made:

- The failures of stations and bus are not considered.
- There are no limits on the queue sizes and therefore no blocking occurs.

For this purpose, CAN is simulated using a simulation language called SLAM II.

The general conclusion drawn from this work was that the asymmetry of traffic loads, inter arrival time variability and basis arrival rates, all will impact the performance of the network with respect to average message delay and average bus utilization. With higher levels of asymmetry of traffic loads, inter arrival time variability between two messages and basis arrival rates the average bus utilizations become larger [Leu94]. It is concluded in this research work that the non-linear effects of asymmetry of traffic loads, inter arrival time variability and basis arrival rates will significantly affect average message delays. After going through this research work we can conclude that the asymmetry of traffic loads will affect the behavior of CAN in terms of average message delay.

CHAPTER 3
SIMULATION MODEL

This CAN simulation model can contain up to 63 stations and each station can send messages with different priorities to the CAN bus. So there will be more priorities than the number of stations. The time of a message transmission is the length of message multiplied by the unit transmission time. For J1939, the baud rate is 250 kbps, so the unit transmission time will be 4 microseconds per bit [Leu94]. In this model, if there is any error in transmission of a message, it will be retransmitted. Bit errors are generated at random points. Figure 3.1 below shows the CAN model used for this simulation.

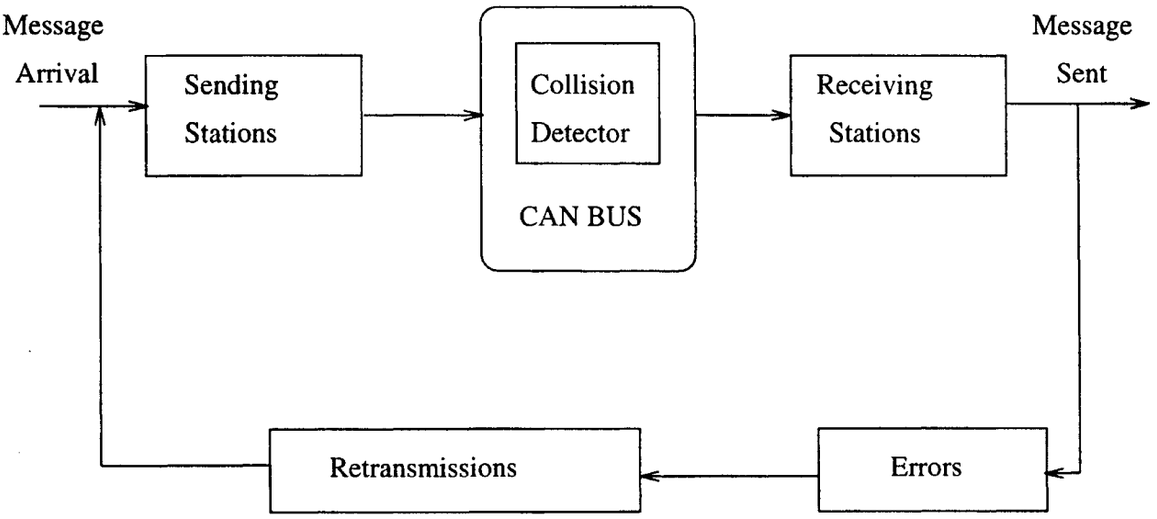

Figure 3.1 CAN Model [Leu94]

3.1 Assumptions

For the simulation, we make the following assumptions:

- Stations and bus failures are not considered.
- The transmission speed is 250 kbps.
- Message arrival is uniformly distributed or exponentially distributed.
- Errors are generated at random points in times.

3.2 CAN Protocol

CAN uses a serial-communication protocol with three layers. They are the Physical Layer, Data Link Layer, and Application Layer as shown in the Figure 3.2 below.

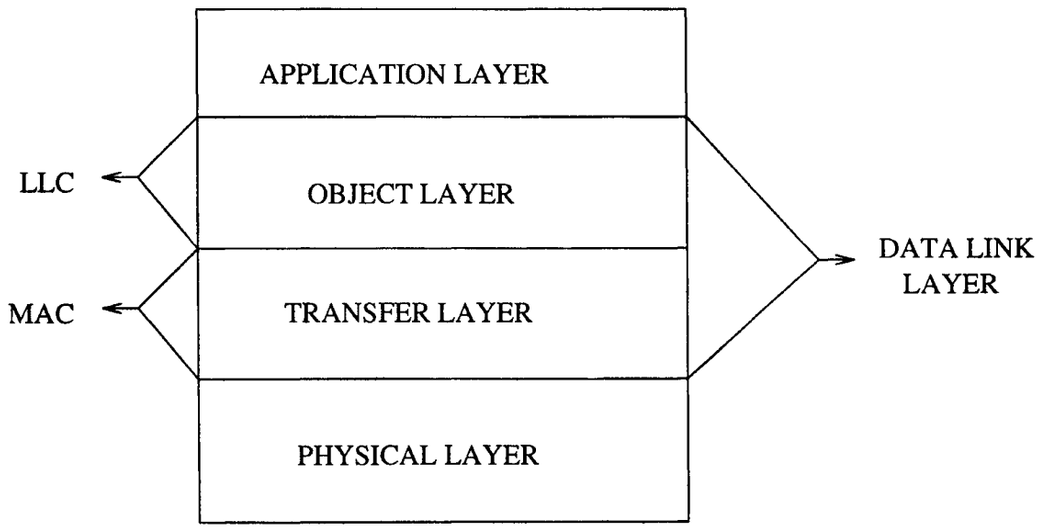

Figure 3.2 Layered Structure of a CAN Node

The Physical Layer performs bit level functions like signal level representation, bit representation and acts as a transmission medium. The Data Link Layer is further sub-divided into the Medium Access Control (MAC) sublayer and Logical Link

Control (LLC) sublayer. The MAC sublayer performs message level functions like message filtering, message handling and status handling. The LLC sublayer performs object level functions like error detection, message validation, arbitration, message framing, transfer rate, timing, acknowledgment, error detection and signalling. The scope of the LLC layer includes deciding which messages received by the MAC are to be used, determining which messages are to be transmitted and providing an interface to the host CPU [Pha88]. MAC is also known as the Transfer Layer and LLC is also known as Object Layer [Gmb91]. The Transfer Layer is the kernel of CAN protocol.

A protocol known as the bus arbitration protocol is used to resolve the collisions by arbitration. The mechanism of arbitration guarantees that neither information nor time is lost. When a Data Frame and a Remote Frame with the same Identifier are initiated at the same time, the Data Frame prevails over the Remote Frame. Arbitration is an important feature of CAN that allows for non-destructive collision detection. When a station listens to a dominant '0' bit on the bus and sends a recessive '1' bit, it backs off and loses the bus arbitration. After all arbitration bits are sent, the winner holds the bus.

3.3 Requirements of CAN

A Controller Area Network should have prioritization of messages, modular design, configuration flexibility, error detection and error handling schemes.

3.3.1 Prioritization of Messages

Since operating schedules are tied to their respective processes, messages in a real-time network like CAN are generated at each node. When two messages are generated at the same time, a conflict occurs. In such occasions the standard arbitration schemes such as first-come first-serve or round-robin will not identify the urgency of a message. Therefore, in this CAN simulator a bitwise arbitration takes place and a message with

urgency obtains access of the bus.

3.3.2 Modular System Concept

An overall optimization layer should be added to enhance the performance of a car. By dividing the entire system into manageable parts the whole system can be efficiently managed.

3.3.3 Configuration Flexibility

CAN must be flexible enough to add or delete any number of stations without affecting the underlying hardware or software and Application Layer.

3.3.4 Error Detection

The CAN Protocol must provide a powerful error detection mechanisms to ensure a low rate of undetected errors. For detecting errors, stations compare the bits transmitted with the bits detected on the bus. Stations also use a Cyclic Redundancy Check (CRC), Bit Stuffing and a Message Frame Check for detecting errors. This Error Detection of CAN can detect all global errors, all local errors, up to 5 randomly distributed errors in a message, burst errors of length less than 15 in a message and any odd number of errors in a message. These errors include bit errors, stuff errors, CRC errors, form errors and acknowledgement errors. A bit error can be detected by a transmitter if the bit value on the bus is different from the bit value it sent. A stuff error can be detected by detecting six consecutive bits. These six consecutive bits violate the property used for bit stuffing. A CRC error is detected when the CRC computed by the transmitter doesn't match the CRC computed by the receiver. An acknowledgement error is detected when a dominant bit is not read in the ACK slot field. A form error is detected when a fixed bit field has an incorrect bit. According to Robert Bosch GmbH, the rate of undetected errors is below 4.7×10^{-11}.

3.3.5 Error Handling

CAN is able to retransmit a message in case of errors. This will not load the CPUs with transmission control.

3.4 Message Format

The message format in a CAN comprises of Start of Frame, Arbitration field, Control field, Data field, CRC field, Acknowledgment field and End of Frame field. Figure 3.3 below shows this message format used in CAN.

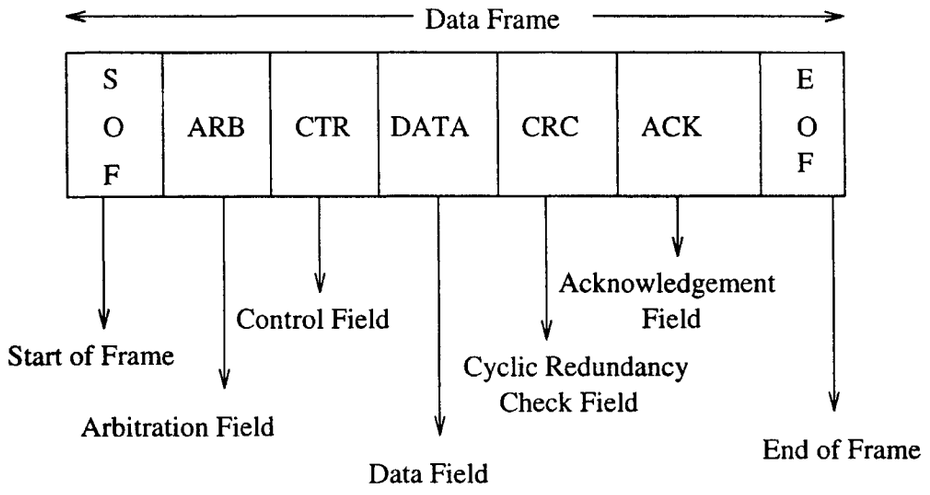

Figure 3.3 Message Format of CAN [Ste88]

3.4.1 Start of Frame

This marks the beginning of a message. This field is a single dominant bit.

3.4.2 Arbitration Field

This consists of the message's identifier and one additional control bit. This field is 11 or 29 bits long. For a J1939 this field is of 32 bits. This field has 11 bits as

Identifier bits, 18 bits of Identifier Extension and SRR, IDE and RTR one bit each. The Identifier field consists of 3 bits of priority code, 1 bit reserved, 1 bit of data page and 6 bits of data content. The Identifier Extension has 2 bits of data content, 8 bits of Protocol Data Units (PDU) which are specific and 8 bits of source address. RTR stands for Remote Transmission Request. RTR is dominant for a Data Frame and recessive for a Remote Frame. The data content field is used to label the data that is contained in the data field of the message.

3.4.3 Control Field

This field contains all necessary control bits. This field has 6 bits. It includes the Data Length Code (DLC) and two bits reserved for future expansion. The reserved bits must be dominant [Gmb91].

3.4.4 Data Field

This contains the original data to be transmitted which can be from 0 to 64 bits. This is used to convey the data associated with the data content label. If the data to be conveyed is more than 64 bits, then it is conveyed across the network through a transport protocol [Ste88].

3.4.5 CRC Field

This field consists of cyclic redundancy check word for error detection. This field is 16 bits long. These 16 bits include a 1 bit CRC delimiter which is recessive.

3.4.6 ACK Field

This field provides a time slot in which all receiving stations can send back an acknowledgement signal. This field has 2 recessive bits containing an ACK slot and an ACK delimiter.

3.4.7 End of Frame

This field marks the end of a message. This field consists of 7 recessive bits.

The interframe space separates Data Frame and Remote Frames. This contains the bit fields INTERMISSION and BUS IDLE. This interframe space must be more than 3 bits long.

CHAPTER 4

IMPLEMENTATION

The basic objectives of this CAN simulator are to evaluate the performance of CAN under different load constraints. These constraints include the simulation duration, network speed, number of periodic samplings, number of nodes, number of messages at each node, message types, priorities of messages, length of each message, bandwidth, error generation rate, error generation type and random number seed. The output of this simulator includes a summary for each node, a summary for all nodes. Each summary consists of performance measures such as network load, network throughput, average latency time, average turnaround time, number of missed deadlines, number of errors and number of retransmissions. The following sections describes the environment in which this simulator was developed and its implementation.

4.1 Environment

This CAN simulator is implemented on a IBM PC using Microsoft Quick C and Quick Windows for the Graphical User Interface (GUI). The CAN simulator is coded in the C Programming Language and uses the Quick Windows functions for the GUI.

4.2 Implementation

In the simulator, the physical time is mapped to simulation time. After reading the input from both GUI and input file, the initilization of all parameters is performed. Simulation is started with the arrival of the first message. Once the message is released, it continues arriving at regular intervals if it is a periodic message. If the incoming message is sporadic, then it has its own arrival rate. When more than one

message is started at the same time, an arbitration process will be initiated to resolve the conflict. A dominant '0' bit overwrites a recessive '1' bit. The winner in this arbitration process gains access of the bus. If the bus is idle because of no message arrival, a idle time counter is incremented. For transmission of mesages across the CAN, logical bits are used. All the 'k' bit transfer results in additional 'k' bit time being spent in the process of simulation. After the logical bit is transferred, this bit is received by all receivers simulatneously. If more than five consecutive bits of same value are sent, then a bit is stuffed. For this reason, the simulation time is incremented by one. In this way all logical bits are transmitted until the End of Frame or an error occurs.

Errors are generated uniformly using a random number generator. The user can specify the rate at which errors are generated. When there is an error in transmission, the bit being transmitted at that particular time is complemented to produce an error condition. All transmitters and receivers monitor the transmission of bits on the bus. All transmitters can detect bit errors and acknowledgment erros and all receivers can detect CRC errors, stuff errors and frame errors. For a message, extended frame is taken to be the basic data structure. By ignoring the extended message format bits, a standard frame can also be obtained. The first 6 bits are used for priority assignment in the arbitration field. The data in the data frame is generated randomly. This data can consists of 0 to 64 bits.

In in the process of checking the CRC values of the transmitter and the receiver, a 15 bit shift register is simulated to perform polynomial division, and the reminder is the CRC sequence. This value is computed for the bits ranging from beginning of Start of Frame to the end of the Data Field. On the receiver side also, the same calculations are performed. After this they are matched with the CRC sequence of the transmitter. If they don't match, a CRC error is generated. After all bits are sent, the next message is transmitted. If there are no messages, the simulator keeps on

checking for the arrival of any new message until the simulation time ends. Statistics are collected for each node and the performance measures for the whole network for the time window specified by the user.

4.3 Graphical User Interface

The Graphical User Interface (GUI) will help the user to visually set the input parameters. In Figure 4.1 below, GUI is shown. As soon as the simulator is started, a welcoming screen appears for the CAN simulator followed by the main menu screen.

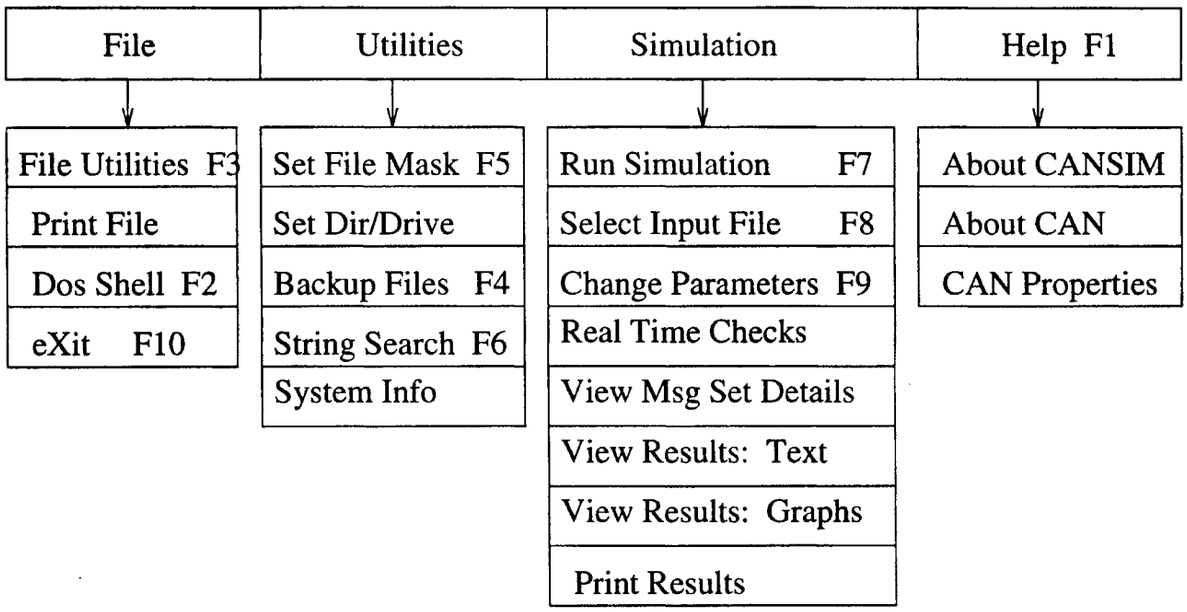

Figure 4.1 Graphical User Interface of CAN

The File Utilities submenu performs the file operations like copying, merging, deleting, renaming, moving, viewing and information of file. The user can print any file by making use of Print File option. From the Dos Shell, user can go to DOS prompt. By making use of Set File Mask option, user can set any file mask he wants. There is a limit of 10 file masks for this simulator. The user can select a directory and a drive for CAN simulation. The user can back up files once the simulation is

finished using any mask for future reference. Also the user can search for a string in the current directory he is working. The user can get information on the system he is working from the System Info option.

The Simulation submenu is the important one for this simulator. Here the user can start the simulation with the default parameters set by just pressing F7 key. Also the user can select a input file for the simulation run by either selecting this option from the submenu or by just pressing F8 key. The user can change the parameters for the simulation by selecting this option or by pressing F9 key. In this option, a user can change the simulation duration in bit times, simulation speed, bandwidth of the network, transmission speed of the network, number of periodic samplings, message arrival mode(periodic or random), error generation rate and seed for the random number generator. The user can check the real-time constraints by selecting this option from this submenu. Also the user can view the message set details like, message name, message arrival time, message length and some other information. Here the user have option for looking at the results both in texual and graphical forms. In the last, the user can print the results on a printer.

The Help submenu contains details about the CAN simulator. The GUI is implemented using Quick Windows. Searching a String involves searching all of the files in that particular directory using fscanf() function. View in the File Utilities is implemented by making use of 'browser'. This browser utility shows the procedures for scrolling a screen up and down on the bottom of the screen. For showing graphs, all of the data is stored in arrays and they are used by Quick C to plot the graphs. For showing graphs, graphic utilities of Quick C are used.

CHAPTER 5

PERFORMANCE ANALYSIS

The metrics used for the performance analysis include:

- Network Load
- Network Throughput
- Average Response Time
- Average Latency Time
- Number of Errors
- Number of Collisions

The simulator was tested with varying input conditions. For verification of the simulator, a single node with a single message is tested with a run of 100 msec, i.e. 25,000 bit times. This message's period is 20 msec and has an extended data frames with 8 bytes of data. This 100 msec run produced the following results:

Total number of messages transmitted = 5

Idle Time = 97.41 msec

Busy Time = 2.59 msec

Error Time = 0 msec

Network Load = 2.59 percent

Network Throughput = 50 msgs/sec

Now let us analyze this message transmission and compare the results with the simulation results. A message with a period of 20 msec in a 100 msec run arrives 5 times. Therefore, the total number of messages transmitted are 5. The network

throughput is calculated as follows: 5 messages arrive per 100 msec, that gives a throughput of $5/100 \times 10^{-3}$ messages/sec., i.e., 50 messages/sec. The total simulation time is the sum of busy time and idle time. The number of bits in message total 128. Therefore, the transmission time is 128 x 4 microsec x 5 messages/1000, which is equal to 2.56 msec. Therefore, the network load is 2.56 percent, which is almost equal to the 2.59 percent calculated from the simulation.

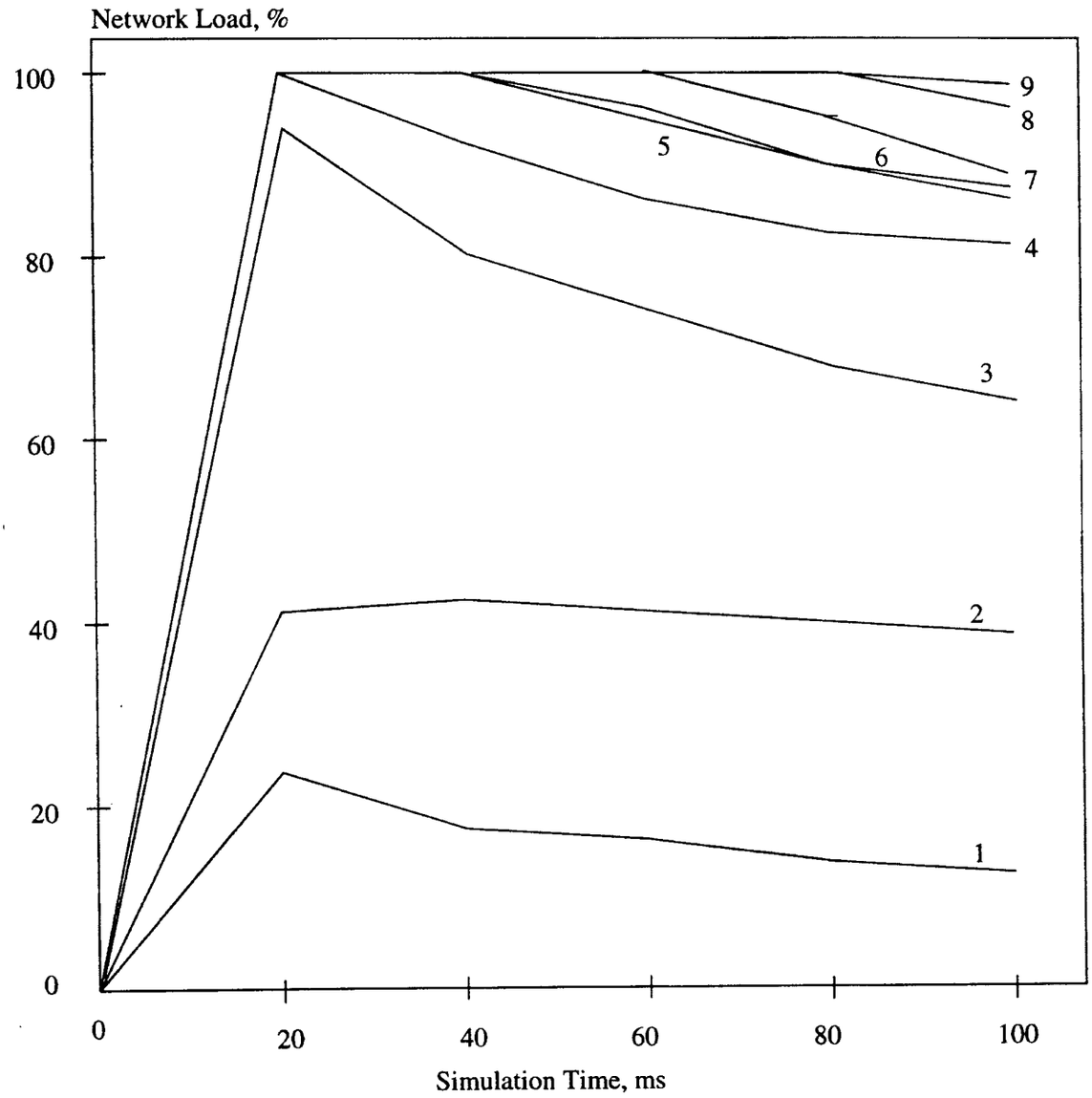

Figure 5.1 Network Load Characteristics

From the above discussion we verified our simulator with a single node and a single message. This simulator is run for 9 different input message sets, with 2, 5, 13, 20, 30, 35, 39, 44, and 50 nodes corresponding to 4, 10, 20, 28, 40, 45, 49, 54, and 60 messages respectively. These message set conditions are labelled as 1, 2, 3, 4, 5, 6, 7, 8, and 9 in the graphs shown in this chapter. Also, these input files are listed in Appendix E. A graph is drawn between the network load and simulation time which is shown in Figure 5.1.

Network Load is defined as the ratio of utilized bus time to the total time i.e., the percentage of time the network is in use.

Network Load = Utilized Time/Total Time

Total time is the simulation run time and utilized time includes both the busy time and error frame time. In the beginning all the messages are released simultaneously by all stations in the network. This is the reason why the load is high at the beginning. But as the simulation proceeds, the load decreases and later enters into a more stable state.

The Time characteristics are shown in Figure 5.2. Here, the three parameters are considered with respect to the network load. They are Busy Time, Idle Time and Error Time. At any point on this graph, the sum of Busy Time, Idle Time and Error Time is nothing but the total simulation time. From this graph we can observe that as the network load is increasing, the idle time is decreasing linearly. This signifies the fact that as bus load increases the busy time increases thereby reducing the idle time. Busy Time increases almost linearly with load. The Error Time is the error frame transmission time. The uneveness in the error time is because of random generation

of errors during simulation.

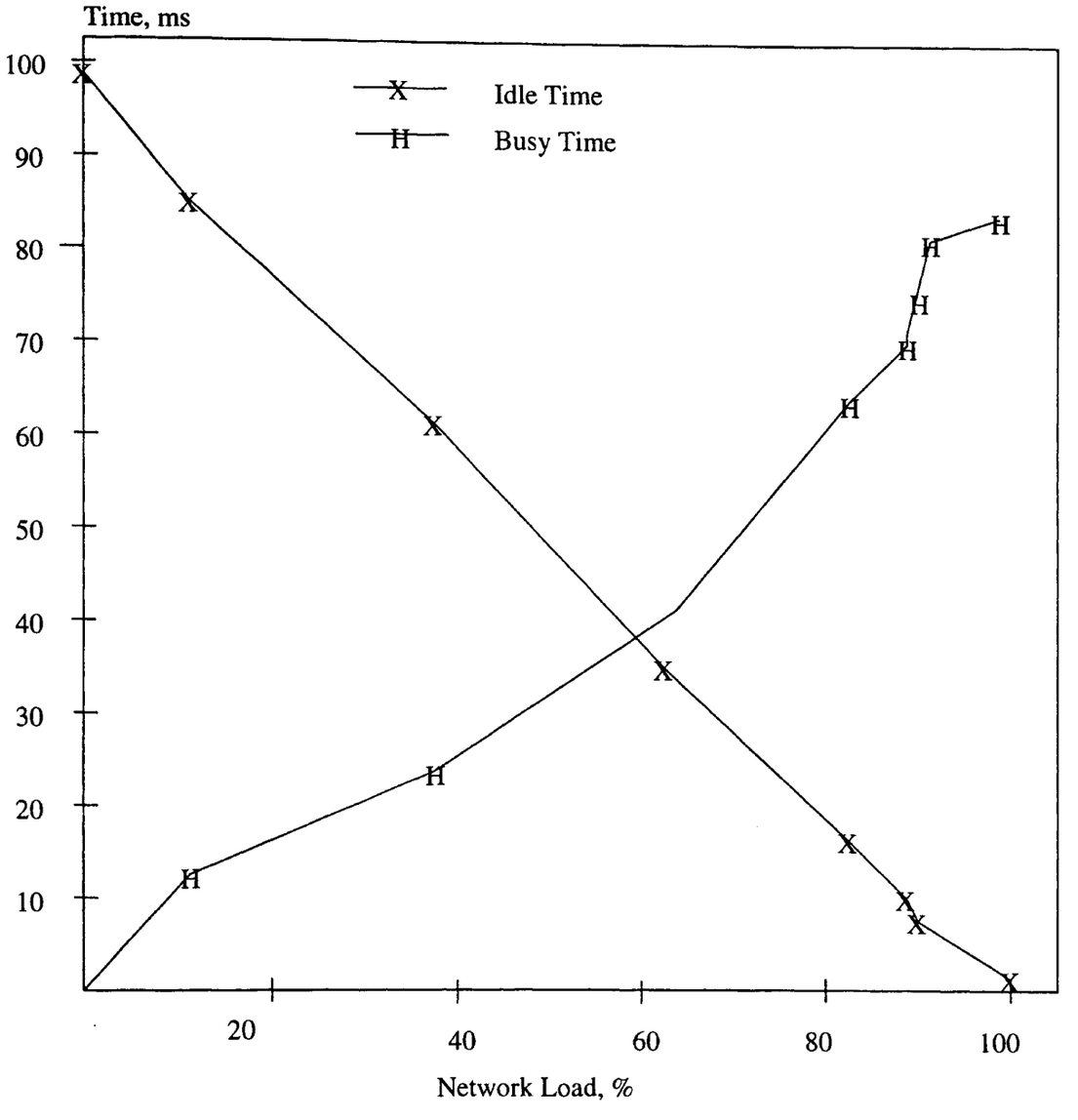

Figure 5.2 Time Characteristics

Network Throughput is defined as the total number of valid messages transmitted per second.

Network Throughput = Total number of messages transmitted/Total Time

A graph is drawn between Throughput versus Load. As expected, we can observe a linear increase in the throughput as the load increases. There is an increase

in throughput after the second and third load points. This is because as the bus gets busy with messages, there is a dramatic increase in the number of messages transmitted per second.

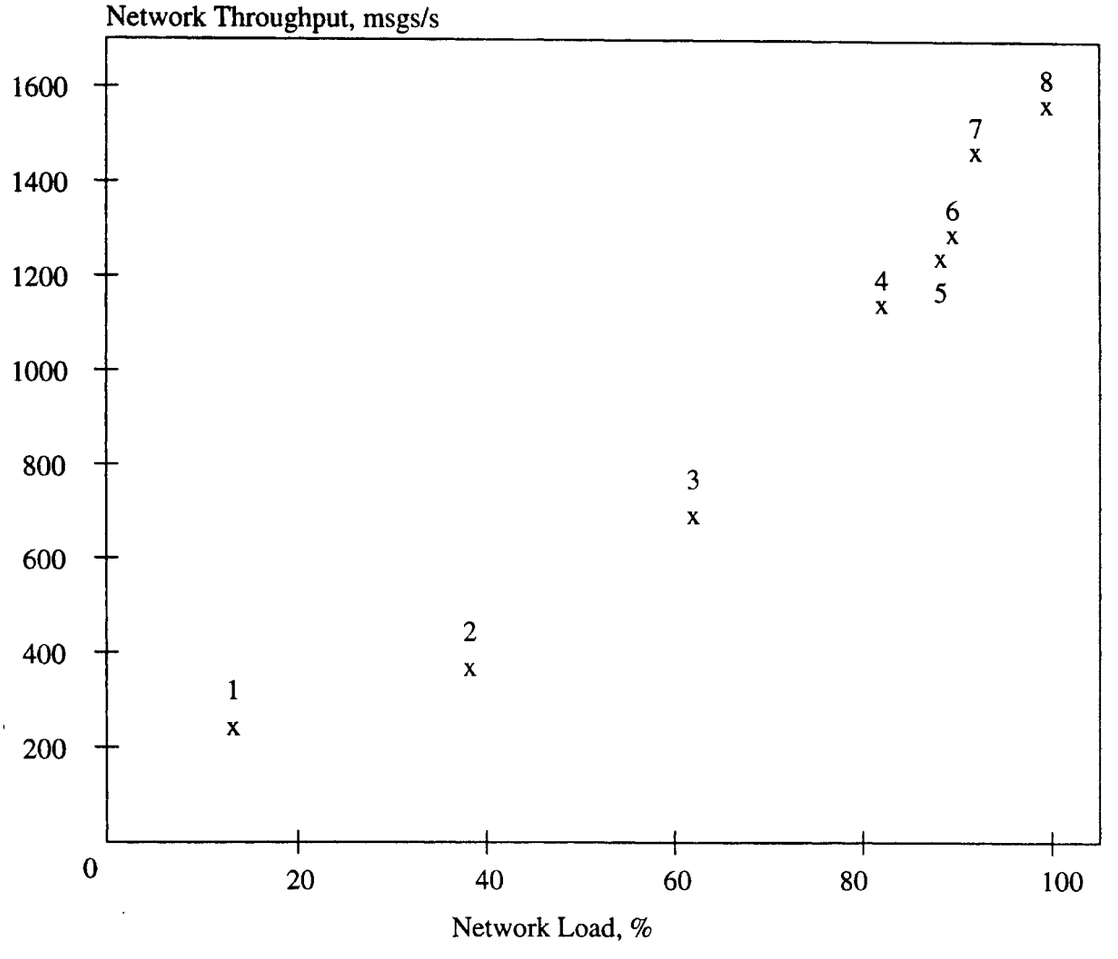

Figure 5.3 Throughput Characteristics

The average response time is the average amount of time taken to gain the access of the bus from the time they are generated. The graph in Figure 5.4 shows this characteristics. Fewer stations with few message sets produce very low response time. But as the number of stations and message sets grow in number there is a increase in response time also. When the number of stations is 30, there is a sharp increase in the response time. For more than 30 stations, the initial response time is

around 7.5 msec. As the simulation time increases the response time decreases and finally reaches a stable state. This signifies the fact that on the whole the average resonse time becomes steady as the simulation time increases.

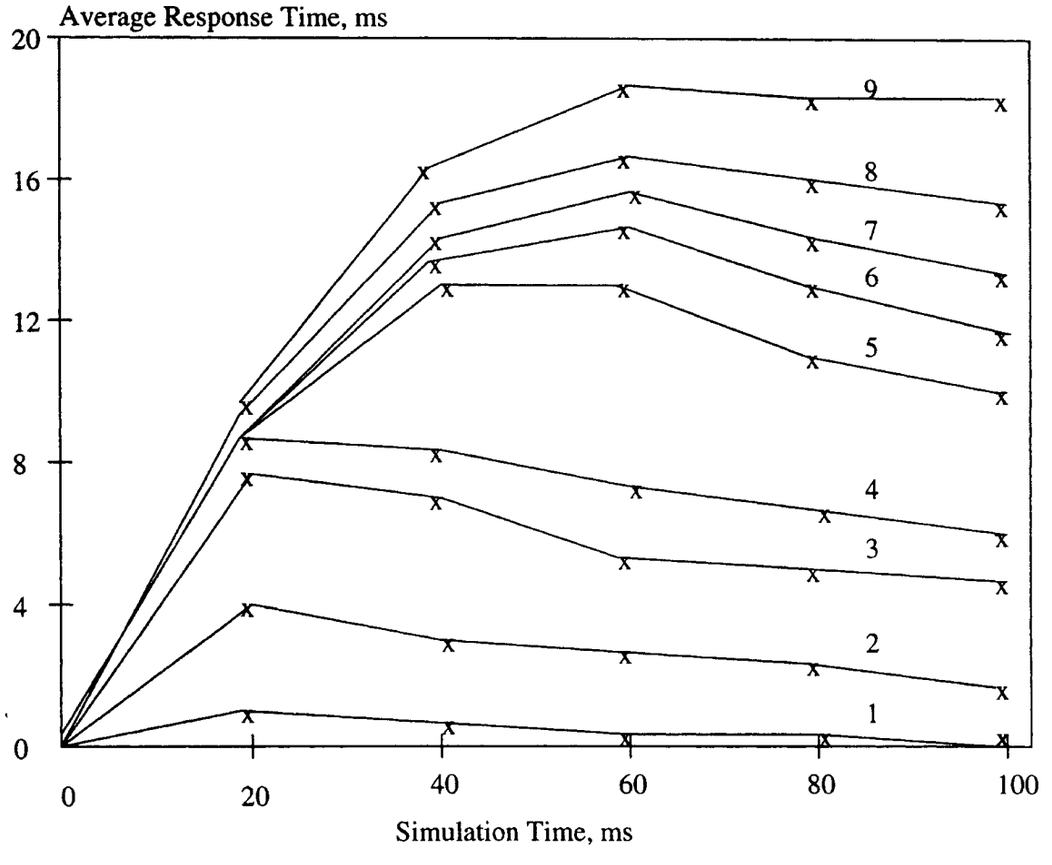

Figure 5.4 Response Time Characteristics

The average latency time is defined as the average time elapsed between message deadline and the actual time of transmission. This is defined for a message that misses its deadlines. From the graph shown in Figure 5.5, we can see that the first three load conditions have zero latencies. This signifies the fact that when the number of

stations are few, no message misses its deadline initially.

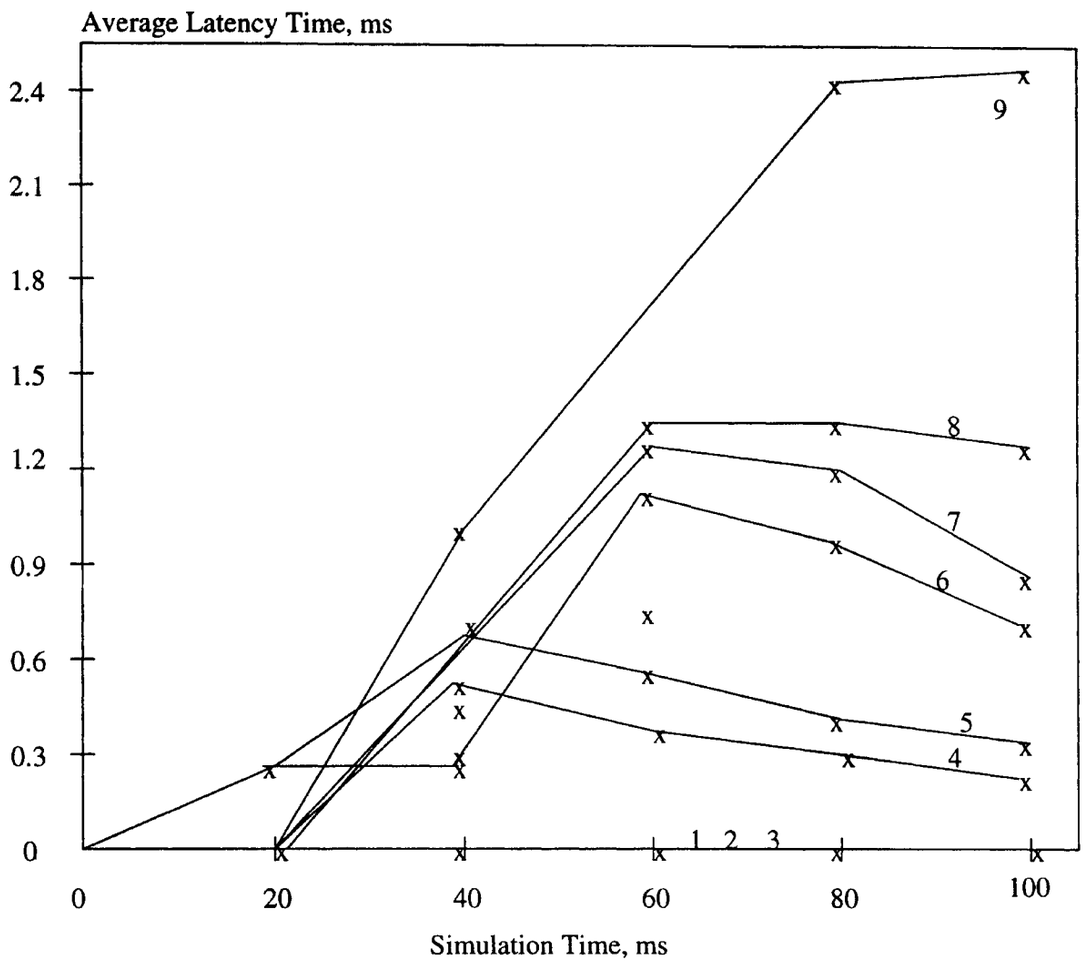

Figure 5.5 Latency Time Characteristics

But as the number of messages and stations grew in number, we can see that there is an increase in the latency time for the messages which are missing their deadlines. This is obvious from that fact that as the load increases on the CAN bus, intially some of messages misses their deadlines contributing to the hike in latency time. After sometime this latency time decreases as the intial load on the bus is accommadated. In this graph of Figure 5.5 the curve numbered 9 has more latency time as it has more number of messages and stations.

The graph in Figure 5.6 shows the variation in the number of collisions as the load is varied. The number of collisions are increased for loads more than 35 percent.

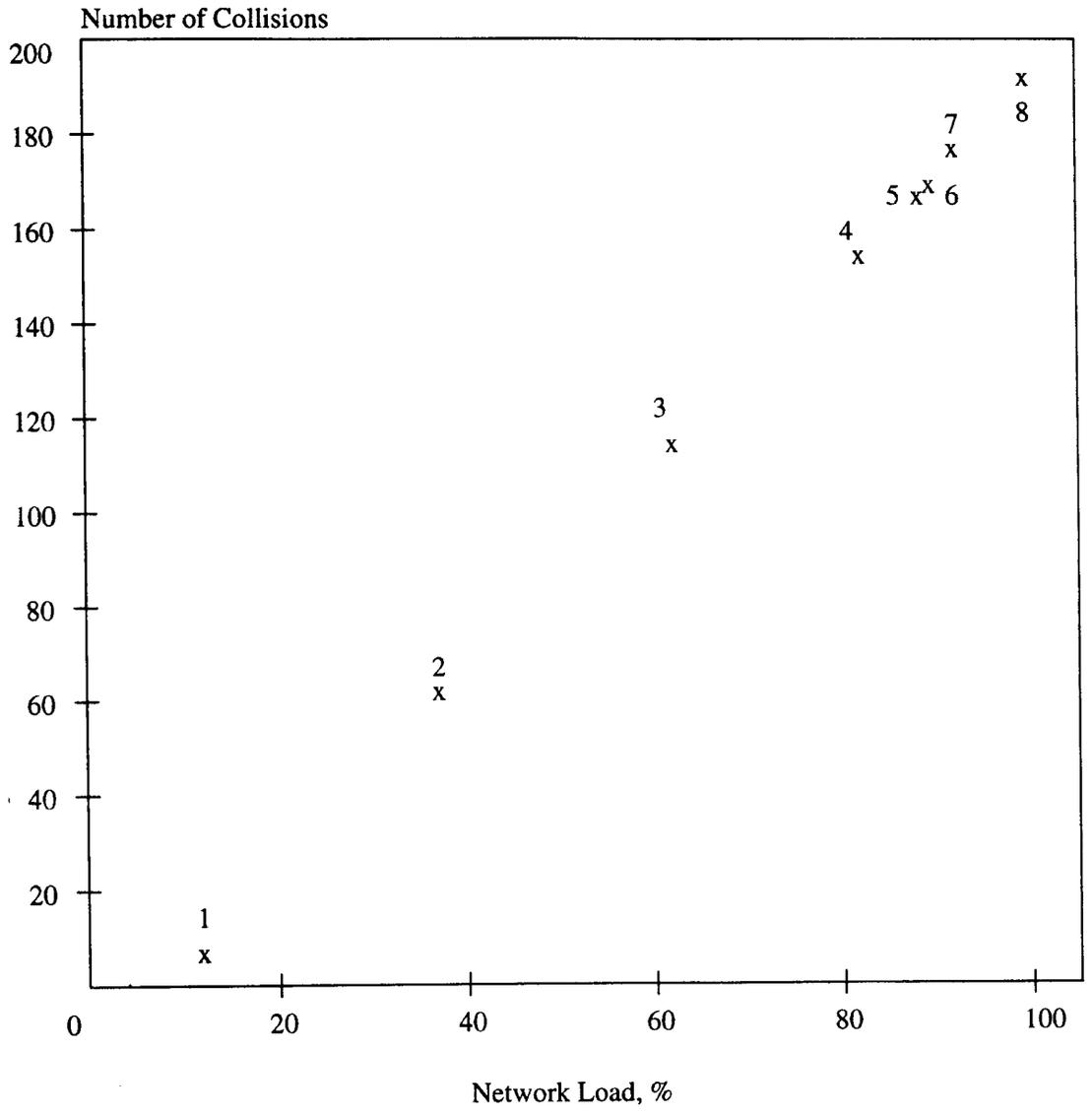

Figure 5.6 Collision Characteristics

On the whole CAN performs well under 40 percent of loads at which a throughput of 400 messages per second is achieved.

CHAPTER 6
CONCLUSION

6.1 Summary

This simulation tests use loading conditions from 0 to 100 percent. Most of the results show good performance for loads below 35 percent of capacity. The throughput at this load is around 320 messages per second. Both periodic and random errors are generated to check the fault tolerance of the network under different loading conditions. On the average for all load points, this simulation produced 29 errors over a 100 msec run. The latency time for loads below 40 percent is zero. The 9th load point has produced more number of messages which missed their deadlines. Around 35 percent of load on the network, there is a sharp increase in the number of collisions. Also the response time is lower for low number message sets and is around 7.5 msec for message sets with more than 30 stations. It is clear that CAN enhances its performance by not destructing messages due to collisions. This feature saves time compared to other protocols where the messages are destructed due to collisions.
On the whole CAN performs well under 40 percent of loads for which a throughput of around 400 messages per second is achieved.

6.2 Future Work

As CAN still a developing network protocol, more and more research has to be done in this area. One could study the fault tolerance of CAN by generating more errors. Some more research could be done to implement this simulator in a Windows environment.

BIBLIOGRAPHY

[Ey89] Horst Ey. Controller Area Network (CAN) components. *Electronic Components and Applications*, 1989.

[Fis92] Paul A. Fishwick. Getting started with Simulation Programming in C and C++. In *ACM Transactions on Modeling and Computer Simulation*, 1992.

[Gmb91] Robert Bosch GmbH. Can version 2.0b. In *Bosch*, 1991.

[Ive88] Wesley. R. Iversen. Intel gets a jump on the Auto Multiplex Market. *Electronics*, 1988.

[Jur86] R.K. Jurgen. Coming from Detroit: Network on Wheels. In *IEEE Spectrum*, 1986.

[KL86] Dais S Kiencke, U. and M. Litschel. Automotive Serial Controller Area Network. In *International Congress and Exposition*, 1986.

[Leu94] Tsao-Jean Leu. Performance Analysis of a Controller Area Network subject to Asymmetric Traffic Loads. Master's Thesis, Department of Computer Science - Oklahoma State University, 1994.

[Mie86] F.O.R Miesterfeld. Chrysler Collision Detection (C^2D) - A Revolutionary Vehicle Network. In *International Congress and Exposition*, 1986.

[Nat93] Pennathur Nataraj. Bitwise Simulation of Controller Area Network. Master's Thesis, Department of Computer Science - Oklahoma State University, 1993.

[Pha86] D.J. Arnett and F.H. Phail. In-Vehicle Networking - Serial Communications Requirement and Directions. *Society of Automotive Engineers*, 1986.

[Pha88] F.H. Phail. Controller Area Network - An In-Vehicle Network Solution. In *International Summer Meeting of the ASAE.* American Society of Agricultural Engineers, 1988.

[Ste88] Mark R. Stepper. J1939 High Speed Serial Communications - The Next Generation Network for Heavy Duty Vehicles. *Society of Automotive Engineers*, 1988.

[WJJ86] Jack R. Volk and Wayne J. Johnson. A Proposal for a Vehicle Network Protocol Standard. In *Society of Automotive Engineers*, 1986.

[WL92] Wang Zhengou, Huizhu Lu and Marvin Stone. Message Delay Analysis for CAN Based Networks. In Proceedings of the 20th Annual Computer Science Conference, 1992.

APPENDIX A

USING THE SIMULATOR

First, file 'cansim.c' is opened and the project is set to 'cansim.mak'. After this the simulator code is compiled using the 'Build All' option from the Make menu of Quick C. For this we set the libraries and the include paths to the Quick C environment. Once it is compiled, the simulator is ready to go. We can execute the simulator by typing the executable file name, in this case 'cansim' at the 'C:CANSIM' prompt. Once the main program is started, the user sees the menu. First, we can set the parameters of the simulation, including input file name, simulation time in bit time, bandwidth, seed for random number generator, sampling window time in msec, network speed in kbps, number of periodic samplings, error generation mode(periodic or random), periodic error rate, random error rate, simulation speed and sporadic message generation. We can also select an input file from the pull down menu of the simulator.

Once these settings are done, we can start the simulation by pressing F7 or selecting the Run Simulation option from the pull down menu. As the simulation is running we can use buttons 'PAUSE', 'ABORT', and 'STATS' to observe Bus Status, Bus Value and Simulation Clock in the Simulation Control window. In the Simulation Events window, we can see the message transmissions, CRC errors and contention for the CAN bus. We can slow down the simulation speed by entering a large value in the simulation speed input because a larger number here corresponds to slower simulation speed and vice versa.

Once the simulation is finished, we can view the results both in Text and in Graphs mode by choosing the respective options from the Simulation submenu. A user can also make the real-time checks, view the message set details and print the results.

From the Utilities submenu, a user can set mask for files, set directory and drive, backup files in a directory, search for a particular string in that directory and obtain system information.

APPENDIX B

MAKEFILE FOR SIMULATOR

```
PROJ = CANSIM
DEBUG = 1
CC = qcl
CFLAGS_G = /AL /W1 /Ze
CFLAGS_D = /Zi /Zr /Gi  $(PROJ).mdt /Od
CFLAGS_R = /O /Ot /DNDEBUG
CFLAGS = $(CFLAGS_G) $(CFLAGS_D)
LFLAGS_G = /NOI
LFLAGS_D = /INCR /CO
LFLAGS_R =
LFLAGS = $(LFLAGS_G) $(LFLAGS_D)
RUNFLAGS =
OBJS_EXT =
LIBS_EXT = ..\qw\qwamscl.lib
.asm.obj: ; $(AS) $(AFLAGS) -c $*.asm
all: $(PROJ).EXE
cansim1.obj:  cansim1.c $(H)
cansim2.obj: cansim2.c $(H)
funcs1.obj: funcs1.c $(H)
funcs2.obj: funcs2.c $(H)
cansim.obj: cansim.c $(H)
$(PROJ).EXE:
cansim1.obj
cansim2.obj
funcs1.obj
funcs2.obj
cansim.obj
$(OBJS_EXT)
echo > NUL @<<$(PROJ).crf
cansim1.obj +
cansim2.obj +
funcs1.obj +
funcs2.obj +
cansim.obj +
$(OBJS_EXE)
$(PROJ).EXE
$(LIBS_EXT);
<< ilink -a -e "qlink $(LFLAGS) @$(PROJ).crf" $(PROJ)
run: $(PROJ).EXE $(PROJ) $(RUNFLAGS)
```

APPENDIX C

SIMULATOR CODE

```
// cansim.c: This file contains main routine which
// initializes the simulation parameters and sets
// default file as its input file.

#include <qwadv.h>
#include <qwkeys.h>
#include <stdlib.h>
#include "protos.h"
#include "const.h"
#include "gui.h"

void main()
{
int i, err, j;
char *menuscrn, *hbscrn;

qwinit();   // qw initialization
welcome();  // display welcome window
simInit();  // initialize simulation engine

menuscrn = getmem(2000);
menuid = 0;      // Window ID of 0 == Top of screen
barattr = 0x4f;
barrev = 0x1f;
charen = 0x4e;
chardis = 0x48;
options = (PM_NOCLOSE | PM_HKPOS | PM_SETPOS);
err = menuinit(menuid, NMENUS, barattr, barrev,
charen, chardis, menubar, options);
if (err)
printf("\n MenuInit returned [%i] ERROR", err);

// Now setup the individual menu lists
style = SNG_BOX | TRANS_SHADOW;
abdr = 0x30;
aen = 0x31;
adis = 0x38;
ahot = 0x34;
for(i=0; i<NMENUS; i++)
   {
     if(msize[i])
      {
err = menuset(i+1, msize[i], style, abdr,
aen, adis, ahot, menutext[i]);
if(err)
 printf("\n MenuSet returned [%i] ERROR on menu %i", err, i);
```

```
      }
   }
menukey(1, 1, F3_KEY);
menukey(1, 2, F2_KEY);
menukey(4, 0, F1_KEY);
menukey(1, 5, F10_KEY);
menukey(2, 3, F4_KEY);
menukey(2, 1, F5_KEY);
menukey(2, 5, F6_KEY);
menukey(3, 1, F7_KEY);
menukey(3, 2, F8_KEY);
menukey(3, 3, F9_KEY);
err = menuon();
if (err)
    printf("\n MenuOn returned [%i] ERROR", err);

flag = 0;
while(1)
  {
    err = menuget(menuscrn, &menu, &item, &flag, &kb);
    if (flag != 0)
     {
      if(flag == 1)
       {
if(menuisdown())
menuclose(menuscrn);
if(menu == 4)
 showhelp();
if(menu == 1)
 {
  if(item == 1)
   fileutil();
  if(item == 2)
   printFile();
  if(item == 3)
   shell();
  if(item == 5)
    {
     if(menuisdown())
      menuclose(menuscrn);
      menuoff();
      exitmm();
    }
       }
  if(menu == 2)
   {
```

```
      if(item == 1)
        popmasks();
      if(item == 2)
        setDirDrive();
      if(item == 3)
        backup();
      if(item == 5)
        search();
      if(item == 6)
        sysinfo();
      if(item == 9)
        setup();
     }
     if(menu == 3)
      {
       if(item == 1)
        runsim();
       if(item == 2)
        selectInputFile();
       if(item == 3)
        simparm();
       if(item == 4)
        realTimeChecks();
       if(item == 5)
        viewInputDetails();
       if(item == 7)
        viewText();
       if(item == 8)
        viewGraphs();
       if(item == 9)
        printResults();
      }
    }
   flag = 0;
}
     }
} //End of main
```

```
// cansim1.c: This file contains the runsim which runs the
// simulation after reading the input parameters.

#include <search.h>
#include <string.h>
#include <qwkeys.h>
#include "const.h"
#include "protos.h"
#include "defs.h"


int  win1, win2;
char *tscrn1, *tscrn2;
char statfile[14], outfile[14], winstat[14];
int  Flag, Dialnumber, Value;
int  bus_value;

void stopsim();
void inctic();
int  compOrder(ORDER *elem1, ORDER *elem2);

//  This function is the main routine that controls the flow
//  within the program.  It also invokes simulation runs
//  for different message sets from input file named input#.inp

void runsim(void)
{
  srand(sp.seed);  //seed the random number generator
  sys_init();
  if(!get_parm())  //read input file
  return;

  // establish output file names

  strcpy(outfile, sp.inputFile);
 *strchr(outfile, '.') = EOS;
  strcat(outfile, ".out");

  // establish the statistics file

  strcpy(winstat, sp.inputFile);
 *strchr(winstat, '.') = EOS;
  strcat(winstat, ".sts");

 if( (op = fopen(outfile,"w")) == NULL)
  {
```

```
    qwbeep();
    sprintf(inbuf, "Error: %s file could not be created", outfile);
    msg();
    return;
 }

if( (wst = fopen(winstat,"w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: %s file could not be created", winstat);
    msg();
    fclose(op);
    return;
 }

if( (g1 = fopen("graph1.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph1.txt file could not be created");
    msg();
    return;
 }
    fprintf(g1, "Network Load Characteristics");
    fprintf(g1, "\nSimulation Time, ms");
    fprintf(g1, "\nNetwork Load, %%\n");

if( (g2 = fopen("graph2.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph2.txt file could not be created");
    msg();
    return;
 }
    fprintf(g2, "Throughput Characteristics");
    fprintf(g2, "\nNetwork Load, %%");
    fprintf(g2, "\nNetwork Throughput\n");

if( (g3 = fopen("graph3.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph3.txt file could not be created");
    msg();
    return;
 }
    fprintf(g3, "Response Time Characteristics");
    fprintf(g3, "\nSimulation Time, ms");
```

```
    fprintf(g3, "\nAvg. Response Time, ms\n");

if( (g4 = fopen("graph4.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph4.txt file could not be created");
    msg();
    return;
 }
    fprintf(g4, "Collisions Characteristics");
    fprintf(g4, "\nNetwork Load, %%");
    fprintf(g4, "\nCollisions\n");

if( (g5 = fopen("graph5.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph5.txt file could not be created");
    msg();
    return;
 }
    fprintf(g5, "Latency Time Characteristics");
    fprintf(g5, "\nSimulation Time, ms");
    fprintf(g5, "\nAvg. Latency Time, ms\n");

if( (g6 = fopen("graph6.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph6.txt file could not be created");
    msg();
    return;
 }
    fprintf(g6, "Error Characteristics");
    fprintf(g6, "\nNetwork Load, %%");
    fprintf(g6, "\nNo. of Errors\n");

if( (g7 = fopen("graph7.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph7.txt file could not be created");
    msg();
    return;
 }
    fprintf(g7, "Busy Time Characteristics");
    fprintf(g7, "\nNetwork Load, %%");
    fprintf(g7, "\nBusy Time, ms\n");
```

```
if( (g8 = fopen("graph8.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph8.txt file could not be created");
    msg();
    return;
 }
    fprintf(g8, "Idle Time Characteristics");
    fprintf(g8, "\nNetwork Load, %%");
    fprintf(g8, "\nIdle Time, ms\n");


if( (g9 = fopen("graph9.txt", "w")) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error: graph9.txt file could not be created");
    msg();
    return;
}
    fprintf(g9, "Error Time Characteristics");
    fprintf(g9, "\nNetwork Load, %%");
    fprintf(g9, "\nError Time, ms\n");


    tscrn1 = getmem(4000);
    win1 = 0;  //open simulation events window
    wopen(5, 13, 75, 23, 184, 0x17, "Simulation Events", tscrn1,
      &win1);
    finish = (long) sp.bandwidth *  sp.simDuration * sp.networkSpeed;
    sample_period = finish / sp.samples;
    sample_time = sample_period ;
    error_period = sp.pErrorRate * sp.networkSpeed;
    periodic_error = error_period;
    random_error = rand_error();

    wprintf(win1, "The simulation will finish at %-lu bit-times\n",
      finish);
    wprintf(win1, "The sampling period is %d ms\n",
       sample_period/sp.networkSpeed);
    wprintf(win1, "The periodic error rate is %d erros/ms\n",
      error_period);

    node_addressing();
    prior_assign();
    msg_cycl();
```

```
    fclose(wst);
    fclose(op);
    wclose(win1);
    free(tscrn1);
    fprintf(g1, "\n");
    fclose(g1);
    fprintf(g2, "\n");
    fclose(g2);
    fprintf(g3, "\n");
    fclose(g3);
    fprintf(g4, "\n");
    fclose(g4);
    fprintf(g5, "\n");
    fclose(g5);
    fprintf(g6, "\n");
    fclose(g6);
    fprintf(g7, "\n");
    fclose(g7);
    fprintf(g8, "\n");
    fclose(g8);
    fprintf(g9, "\n");
    fclose(g9);

    strcpy(inbuf, "Simulation Done!");
    msg();
    return;
}

void stopsim()
{
  fclose(wst);
  fclose(op);
  strcpy(inbuf, "Simulation terminated due to an error!");
  error();
  wclose(win1);
  free(tscrn1);
  return;
}

//  This function performs initialization of the transmitter
//  packet when new message is created

void packs_init(int S)
{
  memset(&(node[S].packs), 0, sizeof(PACKET));
  return;
```

```
}


//  This function performs initialization of the receiver
//  packet when new message is created

void packr_init(int R)
{
  memset(&(node[R].packr), 0, sizeof(PACKET));
  return;
}


//  This function performs initialization of the simulation
//  parameters when new run is started up

void sys_init()
{
 int i, j, k, temp;

  ones = 0;
  zeros = 0;
  count = 0;
  prv_bit = 1;
  idle_time = 0;
  busy_time = 0;
  response_time = 0;
  slack_time = 0;
  collisions = 0;
  losers = 0;
  latency = 0;
  remote = 0;
  missed = 0;
  error_overhead = 0;
  msg_time = 0;
  errors = 0;
  total_msgs = 0;
  pos = 0;
  tic = 0;
  retransmitted = 0;
  transmitted = 0;
  data_frame = 1;
  standard = 1;
  sample_count = 1;
  overload_count = 0;
  form_count = 0;
```

```
  ack_count = 0;
  crc_count = 0;

  bus = 1;
  bus_flag = IDLE;
  for (i = 0;i < MAX_NODES;i++)
   {
    for (j = 0;j < M_MSGS; j++)
     {
      node[i].msg[j].data_len = 0;
      node[i].msg[j].period = 0;
      node[i].msg[j].release = 0;
      node[i].msg[j].deadline = 0;
      node[i].msg[j].trans_time = 0;
      node[i].msg[j].prior = 0;
      node[i].msg[j].arb_lost = 0;
      node[i].msg[j].error_flag = 0;
     }
    packs_init(i);
    packr_init(i);
    node[i].no_of_msgs = 0;
    for (j = 0;j < MAX_OBJS;j++)
      node[i].object[j] = 0;
    node[i].curr_msg = 0;
    node[i].prv_bus = 1;
    node[i].prv_bus_flag = IDLE;
    node[i].transfer = NO;
    node[i].receive = YES;
    node[i].bit_val = 1;
    node[i].ovrhd_flag = 0;
    node[i].ovrhd_delim = 0;
    node[i].err_flag = 0;
    node[i].err_delim = 0;
    node[i].status = ACTIVE;
    node[i].recv_err_cnt = 0;
    node[i].trans_err_cnt = 0;
    node[i].trans_count = 0;
    node[i].lost_count = 0;
    node[i].error_count = 0;
    node[i].dead_count = 0;
   }
}


//  This function gives a random point at which an error may
//  be generated within the simulation.
```

```
int rand_error(void)
{
  int rand_val;

  rand_val = (int) rand() % sp.rErrorRate;
  return(rand_val);
}


//  This function generates addresses for the nodes.

void node_addressing()
{
  int i;
  unsigned char base_address;

  base_address = 0;
  for (i = 0;i < total_nodes;i++)
   {
    node[i].address = base_address + i;
   }
  return;
}


//  This function obtains the simulation input parameters from an
//  input file namely input#, where # gives the order of the file.
//  The parameters include node, and message data such as node name,
//  number of objects, number of messages, message name, message
//  period, message release time, message priority, message data
//  length, type of format (standard/extended), message mode (data/
//  remote).

int  get_parm()
// returns 1 if no problem
// returns 0 if there is a problem
{
 int i, j, num;
 static char buff[82], line[82];

 if(!strlen(sp.inputFile))
   {
     qwbeep();
     strcpy(inbuf, "Input file has not been specified!");
     msg();
```

```
    return 0;
  }

if( (ip = fopen(sp.inputFile,"r") ) == NULL)
 {
    qwbeep();
    sprintf(inbuf, "Error:  Input File %s not read\n\n",
      sp.inputFile);
    msg();
    return 0;
 }

 tscrn1 = getmem(4000);
 win1 = 0;  //open watch window
 wopen(5, 3, 75, 22, 184, 0x17, "Now Reading Input File", tscrn1,
      &win1);
 total_nodes = 0;
 i = 0;
 wprintf(win1, "Here are the  message parameters\n");
 while (fgets(line, 80, ip) != NULL)
  {
   sscanf(line,"%s %d",&node[i].node_name,&node[i].no_of_msgs);
   wprintf(win1, line);
   num = node[i].no_of_msgs;
   for (j = 0;j < num;j++)
    {
     fgets(line, 80, ip);
     wprintf(win1, line);
     sscanf(line, "%s %d %d %d %d %d %d",&node[i].msg[j].msg_name,
               &node[i].msg[j].period, &node[i].msg[j].release,
    &node[i].msg[j].prior,  &node[i].msg[j].data_len,
    &node[i].msg[j].msg_format, &node[i].msg[j].msg_mode);

     node[i].msg[j].period = (node[i].msg[j].period * 250);
     node[i].msg[j].release = (node[i].msg[j].release * 250);
    }

   total_nodes++;
   i++;
  }
fclose(ip);
waitevent();
wclose(win1);
free(tscrn1);
return 1;
}
```

```
//  This function receives the bit sent over the CAN bus.  It is
//  implemented in such a way that all nodes receive the message.
//  The receivers detect and signal errors to the transmitter, to
//  initiate a retransmission

int receive(int r_bit, int br, int indr, int nr)
{
  int j;
  unsigned arr_crc = 0, check;

  switch (pos)
   {
    case 0:  // reception of the interframe space bits
     // Overload condition if less than 3 consecutive
    //  recessive bits are sensed
     if (r_bit != 1)
      return(OVERLOAD_ERROR);
     node[nr].packr.isp |= (r_bit << br);
   return(SUCCESS);
  break;

    case 1:  // reception of the SOF bit
     // form violation if a recessive SOF is sent
             if (r_bit != 0)
              return(FORM_ERROR);
             node[nr].packr.sof |= (r_bit << br);
             return(SUCCESS);
             break;

    case 2:  // reception of the arbitration bits
             node[nr].packr.arb[indr] |= (r_bit << br);
     // remote message sensing
             if (!data_frame && (((standard) &&
                (indr == 2) && (br == 4) && (r_bit == 1)) ||
               ((!standard) && (indr == 0) &&
      (br == 0) && (r_bit == 1))))
               for (j = 0;j < node[nr].no_of_objs;j++)
                if (node[nr].object[j] == node[n].address)
                 {
                  node[nr].msg[j].deadline = tic + 150;
                  return(SUCCESS);
                 }
             return(SUCCESS);
             break;
```

```
case 3:  // reception of the control bits
         node[nr].packr.ctr |= (r_bit << br);
         return(SUCCESS);
         break;

case 4:  // reception of the data bits
         node[nr].packr.dat[indr] |= (r_bit << br);
         return(SUCCESS);
         break;

case 5:  // reception of the CRC sequence bits
         node[nr].packr.crc[indr] |= (r_bit << br);
         // CRC sequence check by receivers
         if ((indr == 0) && (br == 0))
            {
           arr_crc = node[nr].packr.crc[0];
           arr_crc |= (node[nr].packr.crc[1] << 8);
           check = crc_check(nr);
           if (check != arr_crc)
            return(CRC_ERROR);
          }
         return(SUCCESS);
         break;

case 6:  // reception of the acknowledgement bits
         if (br == 0)
           node[nr].packr.ack |= (r_bit << br);
         else {
          // acknowledgement posting by receivers
           node[nr].packr.ack |= (0 << br);
          // fault confinement
           if (node[nr].recv_err_cnt != 0)
            {
             node[nr].recv_err_cnt--;
             if ((node[nr].trans_err_cnt < 128) &&
                (node[nr].recv_err_cnt < 128))
             node[nr].status = ACTIVE;
             }
           }
         // negative acknowledgement detected by the transmitter
         if ((br == 0) && (node[nr].packr.ack != 1))
          {
           return(ACK_ERROR);
          }
         return(SUCCESS);
```

```
            break;

    case 7:  // reception of the EOF bits
             // form violation in EOF bits with
             // the detection of dominant bit
             if (r_bit != 1)
              {
               return(FORM_ERROR);
              }
             node[nr].packr.eof |= (r_bit << br);
             return(SUCCESS);
             break;
    }
 return(OVER); // end of message reception
}



//  This routine is the core of the bit by bit simulation.
//  It represents a dominant bit on the bus with a logical 0,
//  and a recessive bit with a logical 1.  It also performs
//  the job of creating an error condition by complementing
//  the true value at the appropriate error time.

int get_bit(unsigned char g_val, int bits)
{
  if (g_val & (1 << bits))
   {
    if (tic == random_error)
     {
      random_error = tic + rand_error();
      bus_value = 0;
      return(0); // send errorneous bit
     }
    if (tic == periodic_error)
     {
      periodic_error += error_period;
      bus_value = 0;
      return(0); // send errorneous bit
     }
    bus_value = 1;
    return(1); // send the correct bit
    }
  else
   {
    if (tic == random_error)
```

```
     {
      random_error = tic + rand_error();
      bus_value = 1;
      return(1); // send errorneous bit (used to be return(0))
     }
    if (tic == periodic_error)
     {
      periodic_error += error_period;
      bus_value = 1;
      return(1); // send errorneous bit (used to be return(0))
     }
    bus_value = 0;
    return(0); // send the correct bit
   }
}


//  This function transmits a bit over the bus.  The sender
//  station transmits bit by bit.  The sender station also
//  monitors the CAN bus for potential errors during
//  transmission.  Successful transmission of a bits
//  continues unless an error condition is detected.

int transmit(int n)
{
  int i, b, bit_val, bit_flag;
  unsigned char val;

  //  bus is held by the current transmitter
  node[n].prv_bus_flag = BUSY;
  bus_flag = BUSY;

  while (pos <= 8)
   {
    switch (pos)
     {
      case 0:  // transmission of interframe space bits
     val = node[n].packs.isp;
     for (b = 2;b >= 0;b--)
{
 bit_val = get_bit(val, b);
 if ((bit_flag = msg_filter(bit_val,b,0,0)) != SUCCESS)
 return(bit_flag);
}
       pos++;
       break;
```

```
        case 1:  // transmission of SOF bit
                 val = node[n].packs.sof;
                 bit_val = get_bit(val, 0);
           if ((bit_flag = msg_filter(bit_val,b,0,0)) != SUCCESS)
                   return(bit_flag);
         pos++;
                 break;

        case 2:  // transmission of arbitration bits
                 for (i = 3;i >= 0;i--)
                  {
                   val = node[n].packs.arb[i];
                   for (b = 7;b >= 0;b--)
                    {
                     bit_val = get_bit(val, b);
              if ((bit_flag = msg_filter(bit_val,b,i,0)) != SUCCESS)
                        return(bit_flag);
                    }
}
                 pos++;
                 break;

        case 3:  // transmission of control bits
                 val = node[n].packs.ctr;
                 for (b = 5;b >= 0;b--)
                  {
              if ((bit_flag = msg_filter(bit_val,b,0,0)) != SUCCESS)
                        return(bit_flag);
                  }
         pos++;
                 break;

        case 4:  // transmission of data bits
                 for (i = 7;i >= 0;i--)
                  {
                   val = node[n].packs.dat[i];
                   for (b = 7;b >= 0;b--)
                    {
                     bit_val = get_bit(val, b);
              if ((bit_flag = msg_filter(bit_val,b,i,0)) != SUCCESS)
                        return(bit_flag);
                    }
 }
                 pos++;
                 break;
```

```
      case 5:  // transmission of CRC sequence bits
               for (i = 1;i >= 0;i--)
                {
                 val = node[n].packs.crc[i];
                 for (b = 7;b >= 0;b--)
                  {
                   bit_val = get_bit(val, b);
          if ((bit_flag = msg_filter(bit_val,b,i,0)) != SUCCESS)
                     return(bit_flag);
                  }
}
               pos++;
               break;


      case 6:  // transmission of acknowledgement bits
               val = node[n].packs.ack;
               for (b = 1;b >= 0;b--)
                {
                 bit_val = get_bit(val, b);
          if ((bit_flag = msg_filter(bit_val,b,0,0)) != SUCCESS)
                     return(bit_flag);
                }
       pos++;
               break;

      case 7:  // transmission of EOF bits
               val = node[n].packs.eof;
               for (b = 6;b >= 0;b--)
                {
                 bit_val = get_bit(val, b);
          if ((bit_flag = msg_filter(bit_val,b,0,0)) != SUCCESS)
                     return(bit_flag);
                }
       pos++;
               break;

      case 8:  // end of message transmission
               return(OVER);
    }
  }
}


//  This module simulates the bit stuffing function by adding
```

```
//  a bit time whenever 5 consecutive bits of equal value are
//  detected.

void bit_stuff(int bit_rd)
{
  if (bit_rd == 1)
   {
    if (prv_bit == 1)
     {
      ones++;  // track recessive bits
      if (ones > 5)
       {
        msg_time++;
inctic();  // a complement bit is stuffed
ones = 0;
       }
     }
    else
     {
      zeros = 0;
      prv_bit = 1;
     }
   }
  else
   {
    if (prv_bit == 0)
     {
      zeros++;  // track dominant bits
      if (zeros > 5)
       {
        msg_time++;
inctic();  // a complement bit is stuffed

        zeros = 0;
       }
     }
    else
     {
      ones = 0;
      prv_bit = 0;
     }
   }
  return;
}
```

```
//  This function performs message filtering within the CAN
//  nodes. The broadcast bit is sent to nodes that find a
//  match with their communication objects.

int msg_filter(int bit_val, int bm, int indm, int nm)
{
  int bit_read;

  while (nm < total_nodes)
   {
    if (node[nm].receive == YES)
     {
      bit_read = receive(bit_val,bm,indm,nm);
      if (bit_read != SUCCESS)
       {
        node[nm].recv_err_cnt++;
        if (node[nm].recv_err_cnt >= 128)
          node[nm].status = PASSIVE;
        return(bit_read);
       }
     }
    nm++;
   }
  if (!(((pos == 4) &&
    (indm < (8 - node[n].msg[m].data_len))) ||
    ((pos == 2) && (standard) && ((indm < 2) ||
    ((indm == 2) && (bm < 4))))))) {
    msg_time++;
    // global simulation clock that keeps
    // ticking at each bit transmission
    inctic();

    bit_stuff(bit_read);
   }
 return(SUCCESS);
}


//  This function computes the CRC sequence for the frame
//  at the receiving station.  The CRC sequence is generated
//  by a polynomial division algorithm, using a 15-bit
//  shift register.

int crc_check(int nr)
{
  int i, j, k;
```

```
unsigned crc_seq;
unsigned char crc_nxt;
unsigned char nxt_bit;
unsigned char msb_bit;

crc_seq = 0;
msb_bit = 0;
nxt_bit = 0;
msb_bit = (1 & (crc_seq >> 14));
crc_nxt = node[nr].packr.sof ^ msb_bit;
crc_seq <<= 1;
crc_seq &= 0x00007fff;
if (crc_nxt)
  crc_seq ^= 0x4599;
if (standard)
  k = 2;
else
  k = 0;
for (i = 3;i >= k;i--)
 {
  for (j = 7;j >= 0;j--)
   {
    if (node[nr].packr.arb[i] & (1 << j))
      nxt_bit = 1;
    else
      nxt_bit = 0;
    if (crc_seq & (1 << 14))
      msb_bit = 1;
    else
      msb_bit = 0;
    crc_nxt = nxt_bit ^ msb_bit;
    crc_seq <<= 1;
    crc_seq &= 0x00007fff;
    if (crc_nxt)
      crc_seq ^= 0x4599;
    //  accept the first 12 bits for a standard frame
    if ((standard) && (i == 2) && (j == 4))
      break;
    }
  }
for (i = 7;i >= (8 - node[n].msg[m].data_len);i--)
 {
  for (j = 7;j >= 0;j--)
   {
    if (node[nr].packr.dat[i] & (1 << j))
```

```
        nxt_bit = 1;
      else
      nxt_bit = 0;
      if (crc_seq & (1 << 14))
        msb_bit = 1;
      else
        msb_bit = 0;
      crc_nxt = nxt_bit ^ msb_bit;
      crc_seq <<= 1;
      crc_seq &= 0x00007fff;
      if (crc_nxt)
        crc_seq ^= 0x4599;
      }
    }
  crc_seq |= 1;
  return(crc_seq);
}


//  This function performs arbitration during bus contention
//  by more than one message.  The first 6 bits of the
//  arbitration field are used to determine the winner, depending
//  on their priorities.  A dominant overrides a recessive bit
//  during arbitration.  The station that detects the bus value
//  to be different from its bit value backs off from transmission.

int arbitrate()
{
  int i, j, k, b, bus_val = 1, bit_flg, l;
  unsigned char val;

  pos = 0;
  val = node[n].packs.isp;
  for (b = 2;b >= 0;b--)
   {
    node[n].bit_val = get_bit(val, b);
  if ((bit_flg = msg_filter(node[n].bit_val,b,0,0)) != SUCCESS)
     {
      for(l = 0;l < total_nodes;l++)
       if(node[l].transfer == YES)
        node[l].msg[node[l].curr_msg].error_flag= YES;
      return(bit_flg);
     }
   }

  pos++;
```

```
 val = node[n].packs.sof;
 node[n].bit_val = get_bit(val, 0);
if ((bit_flg = msg_filter(node[n].bit_val,0,0,0)) != SUCCESS)
  {
   for (l = 0;l < total_nodes;l++)
     if (node[l].transfer == YES)
       node[l].msg[node[l].curr_msg].error_flag= YES;
   return(bit_flg);
  }

 pos++;

 fprintf(op,"\t(Node, Message) = ");
 wprintf(win1,"\t(Node, Message) = ");
 for (j = 3;j >= 0;j--)
  {
   for (b = 7;b >= 0;b--)
    {
     // all station places their bit values on the bus
     for(i = 0;i < total_nodes;i++)
       if (node[i].transfer == YES)
        {
         val = node[i].packs.arb[j];
         node[i].bit_val = get_bit(val, b);
         bus_val &= node[i].bit_val;
        }

     // every station checks if the bit it placed on the bus
     // is the same as the bit that is being transmitted.

     for(i = 0;i < total_nodes;i++)
       if (node[i].transfer == YES)
        {
   if ((node[i].bit_val != bus_val) && (node[i].bit_val == 1))
          {
           node[i].transfer = NO;
           k = node[i].curr_msg;
           node[i].msg[k].arb_lost = YES;
           node[i].lost_count++;
           fprintf(op,"(%d,%d), ", i, k);
           wprintf(win1,"(%d,%d), ", i, k);
           losers++;
           count--;
          }
         else
          {
```

```
            n = i;
           }
         }
       if((bit_flg=msg_filter(node[n].bit_val,b,j,0)) != SUCCESS)
        {
         for (l = 0;l < total_nodes;l++)
           if (node[l].transfer == YES)
             node[l].msg[node[l].curr_msg].error_flag= YES;
         return(bit_flg);
        }
       bus_val = 1;
      }
    }
  fprintf(op,"lost arbitration at time %ld\n\n",tic);
  wprintf(win1,"lost arbitration at time %ld\n\n",tic);

  // more than one message has the same priority assigned to it

  if (count > 1)
   {
    wprintf(win1, "\nError in priority assignment, Quits\n");
    stopsim();
    return FAILURE;
   }
  return(SUCCESS);
}


//  This function transmits a overload frame when an
//  overload error occurs.

int send_overload_frame()
{
  int i, b, bit_val;
  unsigned char val;

  val = 0x00;
  for (b = 5;b >= 0;b--)      // six overload flags
  {
    bit_val = get_bit(val, b);
    if (bit_val)
      return(FAILURE);
    for (i = 0;i < total_nodes;i++)
      if (node[i].receive == YES)
        node[i].ovrhd_flag |= (bit_val << b);
    inctic();
```

```
    msg_time++;
   }

  val = 0xff;
  for (b = 7;b >= 0;b--)  // eight overload delimiters
   {
    bit_val = get_bit(val, b);
    if (!bit_val)
      return(FAILURE);
    for (i = 0;i < total_nodes;i++)
      if (node[i].receive == YES)
        node[i].ovrhd_delim |= (bit_val << b);
    inctic();

    msg_time++;
   }
 return(SUCCESS);
}


//  This function transmits an error frame when an
//  ACK error, CRC error, form error, or bit error
//  occurs.

int send_error_frame(int ne)
{
  int i, b, bit_val;
  unsigned char val;

  //  transmitter sending error frame
  if ((ne == n) && (node[n].status == ACTIVE))
   {
    node[n].trans_err_cnt += 8;
    if (node[n].trans_err_cnt >= 128) // error passive node
      node[n].status = PASSIVE;
    if (node[n].trans_err_cnt >= 256) // faulty node
      node[n].status = BUS_OFF;
   }
  if (node[ne].status == ACTIVE)
    val = 0x00;
  else
    val = 0x37;

  for (b = 5;b >= 0;b--)  // six error flags
   {
```

```
    bit_val = get_bit(val, b);
    for (i = 0;i < total_nodes;i++)
      if (node[i].receive == YES)
        node[i].err_flag |= (bit_val << b);
    if (node[ne].err_flag != val)
      return(FAILURE);
    inctic();

    msg_time++;
   }
  val = 0xff;
  for (b = 7;b >= 0;b--)  // eight error delimiters
   {
    bit_val = get_bit(val, b);
    //  receiver detects the first bit to be dominant
    if ((!bit_val) && (b == 8))
     {
      node[nr].recv_err_cnt += 8;
      if (node[nr].recv_err_cnt >= 128)
        node[nr].status = PASSIVE;
      return(FAILURE);
     }
    for (i = 0;i < total_nodes;i++)
      if (node[i].receive == YES)
        node[i].err_delim |= (bit_val << b);
    inctic();

    msg_time++;
   }
  return(SUCCESS);
}


//  This module performs the initiation of a message transfer.
//  The message may contain a data frame, remote frame,
//  error frame, or a overload frame.

void msg_transfer(int mode)
{
  int i, k;

  while (mode < 6)
   {
    switch (mode)
     {
      case 0:  // action after a successful message transfer
```

```
fprintf(op,"\tMessage %d of Node %d TRANSMITTED ",m,n);
wprintf(win1,"\tMessage %d of Node %d TRANSMITTED ",m,n);
fprintf(op,"at time %ld\n\n",tic);
wprintf(win1,"at time %ld\n\n",tic);
busy_time += msg_time;
        response_time += (tic - msg_time) - node[n].msg[m].deadline;
        msg_time = 0;
        packs_init(n);
        for (i = 0;i < total_nodes;i++)
         {
           packr_init(i);
         }
        node[n].trans_count++;
        node[n].transfer = NO;
        node[n].prv_bus = 1;
        node[n].prv_bus_flag = IDLE;
        node[n].msg[m].error_flag = NO;
        node[n].msg[m].deadline += node[n].msg[m].period;
        slack_time += node[n].msg[m].deadline - tic;

       // check node status
        if (node[n].trans_err_cnt != 0)
          {
           node[n].trans_err_cnt--;
           if ((node[n].trans_err_cnt < 128) &&
               (node[n].recv_err_cnt < 128))
                node[n].status = ACTIVE;
          }
        if (node[n].msg[m].deadline < tic)
         {
 fprintf(op,"\n\tMessage %d of node %d MISSED deadline by ", m, n);
 fprintf(op,"%ld bit-times\n\n", tic -
 (long) node[n].msg[m].deadline);
 wprintf(win1,"\n\tMessage %d of node %d MISSED deadline by ", m, n);
 wprintf(win1,"%ld bit-times\n\n", tic -
 (long) node[n].msg[m].deadline);

                 node[n].dead_count++;
                 latency += tic - (long) node[n].msg[m].deadline;
                 missed++;
                }

               if (data_frame)
                 transmitted++;
               else
                 remote++;
```

```
                return;

        case 1:  // initiation of a data/remote transfer
                mode = transmit(n);
                break;

        case 2:  // action after an overload error occurs
                for (i = 0;i < total_nodes;i++)
                  packr_init(i);
                retransmitted++;
                errors++;
                overload_count++;
    fprintf(op,"\tOVERLOAD ERROR in Message %d of Node %d ",m, n);
    fprintf(op,"at time %ld\n\n",tic);
    wprintf(win1,"\tOVERLOAD ERROR in Message %d of Node %d ",m, n);
    wprintf(win1,"at time %ld\n\n",tic);

                node[n].error_count++;
                node[n].msg[m].error_flag = YES;
                if (!send_overload_frame())
                 {
   fprintf(op,"\tError in Overload frame at %ld\n\n", tic);
   wprintf(win1,"\tError in Overload frame at %ld\n\n", tic);
                  mode = 3;
                  break;
                 }
                else
                 {
                  error_overhead += msg_time;
                  msg_time = 0;
                  return;
                 }

        case 3:  // action after a form error occurs
                for (i = 0;i < total_nodes;i++)
                  packr_init(i);
                retransmitted++;
                errors++;
                form_count++;
           fprintf(op,"\tFORM ERROR in Message %d of Node %d ",m, n);
   fprintf(op,"at time %ld\n\n",tic);
           wprintf(win1,"\tFORM ERROR in Message %d of Node %d ",m, n);
           wprintf(win1,"at time %ld\n\n",tic);

                node[n].error_count++;
                node[n].msg[m].error_flag = YES;
```

```
                if (!send_error_frame(nr))
                 {
fprintf(op,"\tError in ERROR frame at %ld\n\n", tic);
wprintf(win1,"\tError in ERROR frame at %ld\n\n", tic);
mode = 2; // Error in Error frame
                  break;
                 }
                else
                 {
                  error_overhead += msg_time;
                  msg_time = 0;
                  return;
                 }

      case 4:   // action after a CRC error occurs
                for (i = 0;i < total_nodes;i++)
                  packr_init(i);
                retransmitted++;
                errors++;
                crc_count++;
                fprintf(op,"\tCRC ERROR in Message %d ", m);
       fprintf(op,"of Node %d at time %ld\n\n", n, tic);
                wprintf(win1,"\tCRC ERROR in Message %d ", m);
       wprintf(win1,"of Node %d at time %ld\n\n", n, tic);

                node[n].error_count++;
                node[n].msg[m].error_flag = YES;
                if (!send_error_frame(nr))
                 {
fprintf(op,"\tError in ERROR frame at %ld\n\n", tic);
wprintf(win1,"\tError in ERROR frame at %ld\n\n", tic);
mode = 2; // Error in Error frame
                  break;
                 }
                else
                 {
                  error_overhead += msg_time;
                  msg_time = 0;
                  return;
                 }

      case 5:   // action after a ACK error occurs
                for (i = 0;i < total_nodes;i++)
                  packr_init(i);
                retransmitted++;
                errors++;
```

```
                ack_count++;
        fprintf(op,"\tACK ERROR in Message %d Node %d ",m,n);
fprintf(op,"at time %ld\n\n",tic);
        wprintf(win1, "\tACK ERROR in Message %d Node %d ",m,n);
wprintf(win1, "at time %ld\n\n",tic);

                node[n].error_count++;
                node[n].msg[m].error_flag = YES;
                if (!send_error_frame(n))
                 {
 fprintf(op,"\tError in ERROR frame at %ld\n\n", tic);
 wprintf(win1,"\tError in ERROR frame at %ld\n\n", tic);
 mode = 2; // Error in Error frame
                  break;
                 }
                else
                 {
                  error_overhead += msg_time;
                  msg_time = 0;
                  return;
                 }

      default:
wprintf(win1, "\n Error in message transfer mode, Quits \n ");
stopsim();
return;

    }
  }
}


//  This function generates a CRC sequence for the message.
//  The CRC sequence is computed for the SOF, Arbitration,
//  Control, and Data Fields in that order.

void crc_gen(int Node, int Msg)
{
  int i, j, k;

  unsigned crc_reg;
  unsigned char crc_nxt;
  unsigned char nxt_bit;
  unsigned char msb_bit;

  crc_reg = 0; // initialize shift register
```

```
msb_bit = 0;
nxt_bit = 0;
msb_bit = (1 & (crc_reg >> 14));
crc_nxt = node[Node].packs.sof ^ msb_bit;
crc_reg <<= 1;
crc_reg &= 0x00007fff;
if (crc_nxt)
  crc_reg ^= 0x4599;
if (standard)
  k = 2;
else
  k = 0;
for (i = 3;i >= k;i--)
 {
  for (j = 7;j >= 0;j--)
   {
    if (node[Node].packs.arb[i] & (1 << j))
      nxt_bit = 1;
    else
      nxt_bit = 0;
    if (crc_reg & (1 << 14))
      msb_bit = 1;
    else
      msb_bit = 0;
    crc_nxt = nxt_bit ^ msb_bit;
    crc_reg <<= 1;
    crc_reg &= 0x00007fff;
    if (crc_nxt)
      crc_reg ^= 0x4599;
    // stop after 12 th bit for standard frames
    if ((standard) && (i == k) && (j == 4))
      break;
   }
 }
for (i = 7;i >= (8 - node[Node].msg[Msg].data_len);i--)
 {
  for (j = 7;j >= 0;j--)
   {
    if (node[Node].packs.dat[i] & (1 << j))
      nxt_bit = 1;
    else
      nxt_bit = 0;
    if (crc_reg & (1 << 14))
      msb_bit = 1;
    else
      msb_bit = 0;
```

```
      crc_nxt = nxt_bit ^ msb_bit;
      crc_reg <<= 1;
      crc_reg &= 0x00007fff;
      if (crc_nxt)
        crc_reg ^= 0x4599;
     }
   }
  node[Node].packs.crc[0] = crc_reg & 0x00ff;
  node[Node].packs.crc[1] = (crc_reg & 0xff00) >> 8;
  node[Node].packs.crc[0] |= 1; // crc delimiter
  return;
}


//  This function performs the objective of generating a
//  packet for each message that arrives at a node.  A packet
//  is created in conformance with the frame format in the
//  CAN 2.0 version.  Both standard and extended frames are
//  developed.  The basic structure is that of a extended frame.
//  Standard frames are built over the extended frame.

void packet_gen(int node_no, int msg_no)
{
  int i, j;

  // interframe space consists of 3 recessive bits
  node[node_no].packs.isp |= 0x7;

  // start of frame is a single dominant bit
  node[node_no].packs.sof = 0;

  // first 6 arbitration bits are used for priority
  // following 5 bits are used to represent node address
  node[node_no].packs.arb[3]|=(node[node_no].address >> 3);
  node[node_no].packs.arb[3]|=(node[node_no].msg[msg_no].prior<<2);
  node[node_no].packs.arb[2]|=(node[node_no].address << 5);

  if (standard) {
    if (data_frame)
      node[node_no].packs.arb[2] |= (0x0); // RTR bit dominant
    else  // if remote frame
      node[node_no].packs.arb[2] |= (0x1 <<4); //RTR 12th bit
    node[node_no].packs.arb[2] |= (0x0f);  // 13th onward bits
    node[node_no].packs.arb[1] |= (0xff);
    node[node_no].packs.arb[0] |= (0xff);
    }
```

```
  else { // extended format
    node[node_no].packs.arb[2] |= (0x1 << 4); // SRR bit
    node[node_no].packs.arb[2] |= (0x1 << 3); // IDE bit
    node[node_no].packs.arb[1] &= (0x00); // extended ID to be set
    node[node_no].packs.arb[0] &= (0x00);
    if (data_frame)
      node[node_no].packs.arb[0] |= (0x0); // RTR bit dominant
    else  // if remote frame
      node[node_no].packs.arb[0] |= (0x1); // RTR 32th bit
    }

  for (i = 3;i >= 0;i--)
  // last 4 control bits give the binary value
  // of data length in bytes
  if (standard)
    node[node_no].packs.ctr |= (0x1 << 4); // IDE and r0 bits
  else
    node[node_no].packs.ctr |= (0x0 << 4); // r0 and r1 bits
  node[node_no].packs.ctr |= node[node_no].msg[msg_no].data_len;

  //all data bits are 1's  we dont care what data bytes are
       for (i = 0; i < node[node_no].msg[msg_no].data_len; i++)
node[node_no].packs.dat[i] = 0xff;

  for (i = 7;i >= 0;i--)
    // a 16-bit CRC sequence is obtained
  crc_gen(node_no, msg_no);

  // ack bits are recessive before transmission
  node[node_no].packs.ack |= 3;

  // 7 end of frame bits are recessive
  node[node_no].packs.eof |= 0x7f;

  return;
}

//  This function performs a priority assignment using the
//  rate monotonic priority assignment algorithm.  Higher
//  priorities are assigned to message with smaller periods.
//  The message set is also tested for the two real time
//  constraints before assigning priorities.

void prior_assign()
{
   int i, j, k, l, temp;
```

```
float cost_fn, schedulables;
long p_max, interval_L;

cost_fn = 0;
// test for the first real time constraint
// total cost function is less than unity
for (i = 0;i < total_nodes;i++)
 for (j = 0;j < node[i].no_of_msgs;j++)
   {
     node[i].msg[j].trans_time = (node[i].msg[j].data_len * 8.0 +
                 MAX_OVERHEAD); //bit-times
     cost_fn += node[i].msg[j].trans_time / node[i].msg[j].period;
     total_msgs++;
   }
wprintf(win1, "Cost Function is %f\n", cost_fn);
if (cost_fn >= 1.0)
   {
    wprintf(win1, "No schedule for this message set, Quits\n\n");
    stopsim();
    return;
   }

   // message ordering by message periods
       l = 0;
       for (i = 0; i < total_nodes; i++)
        {
for (k = 0; k < node[i].no_of_msgs; k++)
 {
  order[l].msg = k;
  order[l].node = i;
  order[l].dead = node[i].msg[k].period;
  order[l].trans = node[i].msg[k].trans_time;
  l++;
 }
        }

       qsort( (void *) order, total_msgs, sizeof(ORDER), compOrder);

       for (i = 0;i < total_msgs;i++)
       {
        j = order[i].node;
        k = order[i].msg;
        node[j].msg[k].prior = i;
       }

max_period = order[total_msgs-1].dead;
```

```
      min_period = order[0].dead;

      // test for second real time constraint
      // no inserted idle time

      p_max = order[total_msgs-1].dead;
      interval_L = p_max - 10;
      schedulables = order[total_msgs-1].trans;
      for (i = total_msgs - 2;i >= 0;i--)
       schedulables += floor(((interval_L-1)/order[i].dead))
*order[i].trans;
        if (interval_L < schedulables)
{
 wprintf(win1, "No schedule for this message set, Quits\n\n");
 stopsim();
 return;
}
}



//  This function performs the message cycle.  It checks
//  for new message arrivals, initiates arbitration if more
//  than one message has arrived, then calls the message
//  transfer routine to transmit the message.

void msg_cycl()
{
  int i, j, k, l;
  static unsigned long int next_arrival = 0;
  int mode, filter;
  long int next;
  int IS_THERE_A_MSG;

  bus_flag = IDLE;
  transmitted = 0;
  pos = 0;
  // initial message deadlines are their release times
  for (i = 0;i < total_nodes;i++)
    for (j = 0;j < node[i].no_of_msgs;j++)
      node[i].msg[j].deadline = node[i].msg[j].release;

  // cycle until end of the simulation run

  //open simulation control window
  tscrn2 = getmem(4000);
```

```
  win2 = 0;
  wopen(5, 3, 75, 12, 184, 0x17, "Simulation Control", tscrn2,
&win2);

  // define dialog items
  dialoginit(win2, 3, 1);
  wcolor(win2, 0x17);
  pushbutton(1, 11, 1, 0, 0x17, 0, "  Pause  ", 0x30);
  pushbutton(2, 31, 1, 0, 0x17, 0, "  Abort  ", 0x30);
  pushbutton(3, 51, 1, 0, 0x17, 0, "  Stats  ", 0x30);

  tic = 0; //initialize the clok to zero
  while (tic <= finish)
   {
    // check each node for message arrivals
    for (j = 0;j < total_nodes;j++)
     {
      next = finish;
      IS_THERE_A_MSG = NO;
      for (l = 0;l < node[j].no_of_msgs;l++)
        if (node[j].msg[l].deadline <= next)
         {
          IS_THERE_A_MSG = YES;
          k = l;
          next = node[j].msg[l].deadline;
         }

      // process each node message
      if ((IS_THERE_A_MSG == YES) &&
          (node[j].msg[k].deadline <= tic))
       {
        node[j].transfer = YES;
        node[j].curr_msg = k;
        if (node[j].msg[k].msg_mode)
          data_frame = 1;
        else
        data_frame = 0; // remote request
        if (node[j].msg[k].msg_format == 1)
          standard = 1;
        else
          standard = 0; // extended frame format

// generate a packet if message is already not there
        if((node[j].msg[k].arb_lost != YES) &&
(node[j].msg[k].error_flag != YES))
         packet_gen(j, k);
```

```
        n = j;
        m = k;
count++;  // number of messages arrivals
wprintf(win1, "\n\tAt %lu: (Node, Msg) (%d, %d)
  contends for bus\n", tic, n, m);
      } // process each node message
     } // here ends check for each and every message
    msg_time = 0;
    mode = 1;

    // arbitrate to resolve bus contention
    if (count > 1)
     {
      collisions++;
      mode = arbitrate();
      m = node[n].curr_msg;
      pos = 3;
      wprintf(win1, "\n\tAt %lu: (Node, Msg) (%d, %d)
                  wins arbitration\n", tic, n, m);
     }
    node[n].msg[m].arb_lost = NO;

    if (node[n].msg[m].msg_format == 1)
      standard = 1;
    else
      standard = 0; // extended frame format

    if (node[n].msg[m].msg_mode == 1)
      data_frame = 1;
    else
      data_frame = 0; // remote request

    // no message has arrived, and so bus is idle
    if (count == 0)
     {
      inctic();

      idle_time++;
      if (tic >= periodic_error)
        periodic_error += error_period;
      if (tic >= random_error)
random_error = tic + rand_error();
      count = 0;
     }

    // initiate a message transfer
```

```
    else
     {
      count = 0;
      if ((bus_flag == IDLE) && (node[n].status != BUS_OFF))
       {
        msg_transfer(mode);
        bus_flag = IDLE;
        pos = 0;
       }
     }
   }
   //close simulation control window
   wclose(win2);
   free(tscrn2);
   return;
}


void inctic()
{
 long i;
 float x;

 if (tic == sample_time)
   {
      windowResults();
      sample_time += sample_period;
      sample_count++;
   }

 //monitor simulation control window activity like sim. clock etc.,
wprintfa(win2,  4, 3, 0x17, "Bus status: %s", bus_flag ?
"BUSY" : "IDLE");
wprintfa(win2, 30, 3, 0x17, "Bus value: %s", bus_value ?
"ONE " : "ZERO");
wprintfa(win2,  4, 4, 0x17, "Simulation clock: %-lu bit-times", tic);
wprintfa(win2,  4, 5, 0x17, "Sampling Start Time: %lu msec", sp.t1);
wprintfa(win2,  40, 5, 0x17, "Sampling End Time: %lu msec", sp.t2);


 Flag = 0;
 dialogget(&Dialnumber, &Flag, &Value);
 if(Flag != 0)
   {
     if(Dialnumber == 1)//if pause is selected from control window
       {
```

```
 strcpy(inbuf, "Press Enter or Space bar to resume");
 msg();
      }
   if(Dialnumber == 2) //if abort is selected from control window
      {
sprintf(inbuf, "Simulation Aborted");
tic = finish;
return;
      }
   if(Dialnumber == 3) //if stats is selected from control window
      {
windowResults();
x = (float) tic / sp.networkSpeed;
sprintf(inbuf, "Stats captured at %8.3f ms.", x);
msg();
    }
  }

  i = 0;
  while(i < sp.simSpeed)
  i++;

  tic++;
  return;
}

int compOrder(ORDER *elem1, ORDER *elem2)
{
 if( elem1->dead <  elem2->dead )
 return -1;
 if( elem1->dead >  elem2->dead )
 return 1;
else
 return 0;
}
```

```
// cansim2.c: This file contains input/output functions
// which reads from input file and write to output files.
#include <stdlib.h>
#include <string.h>
#include <qwkeys.h>
#include "const.h"
#include "protos.h"
#include "defs.h"


int win1;
char *tscrn1;
char statfile[14], outfile[14], winstat[14];
int  Flag, Dialnumber, Value;

void readSimParms(void);

void simparm()
{
 int updFlag = 0, saveFlag = 0;
 long bitsimDuration;
 SP temp;
 char *strinp1 =  "             ";
 char *strinp2 =  "             ";
 char *strinp3 =  "             ";
 char *strinp4 =  "             ";
 char *strinp5 =  "             ";
 char *strinp6 =  "             ";
 char *strinp7 =  "             ";
 char *strinp8 =  "             ";
 char *strinp9 =  "             ";
 char *strinp10 = "             ";
 char *strinp11 = "             ";

 memmove(&temp, &sp, sizeof(SP));

 kbsystem(3, DOWN_ARROW_KEY);
 kbsystem(4, UP_ARROW_KEY);
 qwsystem(18, 1);
 qwsystem(12, 32);

  // format simulation parameters data structure for display
  memmove(strinp1, sp.inputFile, sizeof(sp.inputFile));
  sprintf(strinp2, "%8ld", bitsimDuration);
  sprintf(strinp3, "%3d", sp.bandwidth);
  sprintf(strinp4, "%3d", sp.seed);
```

```
sprintf(strinp5, "%5lu", sp.t1);
sprintf(strinp6, "%5lu", sp.t2);
sprintf(strinp7, "%5d", sp.networkSpeed);
sprintf(strinp8, "%2d", sp.samples);
sprintf(strinp9, "%5d", sp.pErrorRate);
sprintf(strinp10, "%5d", sp.rErrorRate);
sprintf(strinp11, "%9lu", sp.simSpeed);

//open dialog window
win1 = 0;
tscrn1 = getmem(4000);
wopen(7, 3, 74, 22, 49, 0x17, "Simulation Parameters", tscrn1,
    &win1);
wprinta(win1, 12, 8, 0x17, "Error generation");
wprinta(win1, 3, 12, 0x17, "Sporadic message generation");

//define dialog items
dialoginit(win1, 19, 1);
wcolor(win1, 0x71);
inputbox(1, 18, 1, "Input file", 0x17, 416, 0, "", strinp1, 0, 0);
inputbox(2, 0, 2, "Simulation duration in bit-time",
    0x17, 410, 0, "", strinp2, 0, 0);
inputbox(3, 18, 3, "Bandwidth", 0x17, 410, 0, "", strinp3, 0, 0);
inputbox(4, 16, 4, "Seed for RNG", 0x17, 410, 0, "", strinp4, 0, 0);
inputbox(5, 9, 5, "Sampling window, t1", 0x17, 410, 0, "", strinp5,
    0, 0);
inputbox(6, 39, 5, "msec, t2", 0x17, 410, 0, "", strinp6, 0, 0);
inputbox(7, 9, 6, "Network speed, kbps", 0x17, 410, 0, "", strinp7,
    0, 0);
inputbox(8, 10, 7, "Periodic samplings", 0x17, 410, 0, "", strinp8,
    0, 0);
radiobutton(9, 29, 8, 0x17, 0, "None", 0x17, 1, 0);
radiobutton(10, 39, 8, 0x17, 0, "Periodic", 0x17, 1, 1);
radiobutton(11, 52, 8, 0x17, 0, "Random", 0x17, 1, 0);
inputbox(12, 9, 9, "Periodic error rate", 0x17, 410, 0, "", strinp9,
    0, 0);
inputbox(13, 11, 10, "Random error rate", 0x17, 410, 0, "",
    strinp10, 0, 0);
inputbox(14, 12, 11, "Simulation speed", 0x17, 410, 0, "", strinp11,
    0, 0);
radiobutton(15, 39, 12, 0x17, 0, "None", 0x17, 2, 1);
radiobutton(16, 52, 12, 0x17, 0, "Random", 0x17, 2, 0);
pushbutton(17, 9, 15, 0, 0x17, 0, "   OK   ", 0x30);
pushbutton(18, 27, 15, 0, 0x17, 0, "  Save   ", 0x30);
pushbutton(19, 47, 15, 0, 0x17, 0, "  Cancel  ", 0x30);
```

```
    // sit in an infinite loop until there is activity
    // in the dialog window
   while(1)
    {
     Flag = 0;
     while(Flag == 0 && Value != ESCAPE_KEY)
     dialogget(&Dialnumber, &Flag, &Value);
     if(Flag == 0 && Value == ESCAPE_KEY)
     break;
     if(Dialnumber >= 17)  //cancel or OK or save
     break;
   }
     wclose(win1);
     free(tscrn1);
     kbsystem(3, 0);
     kbsystem(4, 0);
     qwsystem(18, 0);

     if(Dialnumber == 19) //if cancel is selected
      {
       memmove(&sp, &temp, sizeof(SP));
       return;
      }

      // convert dialog text items to integers or floats
      // and set them to vars
      trim(strinp1);
      strcpy(sp.inputFile, strinp1);
      bitsimDuration   = atol(strinp2);
      sp.simDuration   = bitsimDuration/250;
      sp.bandwidth     = atoi(strinp3);
      sp.seed          = atoi(strinp4);
      sp.t1            = atol(strinp5);
      sp.t2            = atol(strinp6);
      sp.networkSpeed  = atoi(strinp7);
      sp.samples       = atoi(strinp8);
      sp.pErrorRate    = atoi(strinp9);
      sp.rErrorRate    = atoi(strinp10);
      sp.simSpeed      = atol(strinp11);

      if(Dialnumber == 18)
        saveSimParms(); //Save the simulation parameters
        return;
}

void selectInputFile()
```

```
{
char buf[14];

strcpy(buf, po.filemask);
strcpy(po.filemask, "*.inp");
getFileName();
strcpy(sp.inputFile, cfilename);
strcpy(po.filemask, buf);
return;
}
void printResultFile()
{
char buf[14];

strcpy(buf, po.filemask);
strcpy(po.filemask, "*.sts");
getFileName();
strcpy(sp.inputFile, cfilename);
strcpy(po.filemask, buf);
return;
}

//  This function performs a priority assignment using the
//  rate monotonic priority assignment algorithm.  Higher
//  priorities are assigned to message with smaller periods.
//  The message set is also tested for the two real time
//  constraints before assigning priorities.

void realTimeChecks()
{
  int i, j, k, l, temp;
  float cost_fn, schedulables;
  int p_max, interval_L;

  if(!get_parm())
  return;

  cost_fn = 0;
  //test for the first real time constraint
  // total cost function is less than unity
  for (i = 0;i < total_nodes;i++)
    for (j = 0;j < node[i].no_of_msgs;j++)
     {
      node[i].msg[j].trans_time = (node[i].msg[j].data_len*8.0
+MAX_OVERHEAD);
      cost_fn += node[i].msg[j].trans_time / node[i].msg[j].period;
```

```
      total_msgs++;
     }

  win1 = 0;
  tscrn1 = getmem(4000);
  wopen(5, 3, 75, 22, 184, 0x17, "Real-time Constraints", tscrn1,
&win1);

  wprintf(win1, "Cost Function is %f\n",cost_fn);
  if (cost_fn >= 1.0)
    {
      qwbeep();
      wprintf(win1, "\nCost function exceeds 1.0!\n");
    }

   waitevent();
   wclose(win1);
   free(tscrn1);
   return;
}

void viewText()
{
  sprintf(comm, "%s %s", po.browser, outfile);
  system(comm);
  sprintf(comm, "%s %s", po.browser, winstat);
  system(comm);
  return;
}
void viewGraphs()
{
 char* tmp;
 char* dlginfo;
 int   graphNum = 1;

 kbsystem(3, DOWN_ARROW_KEY);
 kbsystem(4, UP_ARROW_KEY);
 qwsystem(18, 1);
 qwsystem(12, 32);

 win1 = 0;
 tscrn1 = getmem(4000);
 wopen(14, 6, 71, 22, 49, 0x17, "Select a graph", tscrn1, &win1);

 dialoginit(win1, 11, 1);
```

```
radiobutton(1, 9, 1, 0x17, 0, "Network Load vs Simulation Time",
                        0x17, 1, 1);
radiobutton(2, 9, 2, 0x17, 0, "Network Load vs Throughput",
                        0x17, 1, 0);
radiobutton(3, 9, 3, 0x17, 0, "Simulation Time vs Response time",
                        0x17, 1, 0);
radiobutton(4, 9, 4, 0x17, 0, "Network Load vs # of Collisions",
                        0x17, 1, 0);
radiobutton(5, 9, 5, 0x17, 0, "Simulation Time vs Latency Time",
                        0x17, 1, 0);
radiobutton(6, 9, 6, 0x17, 0, "Network Load vs Number of Errors",
                        0x17, 1, 0);
radiobutton(7, 9, 7, 0x17, 0, "Network Load vs Busy Time",
                        0x17, 1, 0);
radiobutton(8, 9, 8, 0x17, 0, "Network Load vs Idle Time",
                        0x17, 1, 0);
radiobutton(9, 9, 9, 0x17, 0, "Network Load vs Error Time",
                        0x17, 1, 0);
pushbutton(10, 8, 12, 0, 0x17, 0, "   View   ", 0x71);
pushbutton(11, 31, 12, 0, 0x17, 0, "  Cancel  ", 0x71);
while(1)
{
  Flag = 0;
  while(Flag == 0 && Value != ESCAPE_KEY)
   dialogget(&Dialnumber, &Flag, &Value);
  if(Flag == 0 && Value == ESCAPE_KEY)
   break;
  if(Dialnumber <= 9) //  graph selection
    {
     graphNum = Dialnumber;
    }
  if(Dialnumber == 10) // view graph
    {
    dlginfo = getmem(10000);
    dlgsave(dlginfo);
    mhide();
    tmp = getmem(4000);
    getscrn(1,1,80,25,tmp);
    sprintf(inbuf, "XYGRAPHS.EXE graph%1d.txt", graphNum);
    system(inbuf);
    csoff;
    putscrn(1,1,80,25,tmp);
    free(tmp);
    mshow();
    dlgrestore(dlginfo);
    free(dlginfo);
```

```
   }
  if(Dialnumber == 11) // cancel
   break;
  }
 wclose(win1);
 free(tscrn1);
 kbsystem(3, 0);
 kbsystem(4, 0);
 qwsystem(18, 0);
 return;
}

void printResults()
{
   printResultFile(); //To print a output file
}


void viewInputDetails()
{
   int i, j;
   win1 = 0;
   tscrn1 = getmem(4000);
   wopen(7, 3, 74, 22, 49, 0x17, "Here is the Message Set",
tscrn1, &win1);
   for (i = 0;i < total_nodes;i++)
     for (j = 0;j < node[i].no_of_msgs;j++)
      {
       wprintf(win1, "\n%d %d %s %lu %8lu %5lu %5lu %lu %lu %lu",
        i, j, node[i].msg[j].msg_name,
        node[i].msg[j].data_len, node[i].msg[j].period,
        node[i].msg[j].release, node[i].msg[j].deadline,
    node[i].msg[j].prior,node[i].msg[j].msg_format,
    node[i].msg[j].msg_mode);
      }
      waitevent();


    for(i = 0; i < total_msgs; i++)
     {
      wprintf(win1, "\n%d  %d  %d  %lu  %f", i, order[i].node,
order[i].msg, order[i].dead, order[i].trans);
      if(i % 15 == 0)
      waitevent();
    }
 waitevent();
```

```
 wclose(win1);
 free(tscrn1);
 return;
}

int saveSimParms(void)
{
sprintf(inbuf, "%s\\simparms.dat", startDir);
fp = fopen(inbuf, "wb");
if(fp == NULL)
return 0;
fwrite(&sp, sizeof(SP), 1, fp);
fclose(fp);
return(1);
}

void readSimParms(void)
{
int i;

strcpy(sp.inputFile, "INPUT1.INP");
sp.simDuration  = 25000;
sp.bandwidth    = 1;
sp.seed         = 1;
sp.t1           = 0;
sp.t2           = 100;
sp.networkSpeed = 250;
sp.samples      = 5;
sp.errorGenMode = 2;
sp.pErrorRate   = 1;
sp.rErrorRate   = 1000;
sp.simSpeed     = 70;
sp.sporadicMsgGenMode = 0;
sprintf(inbuf, "The t1 is %lu and t2 is %lu\n", sp.t1, sp.t2);
sprintf(inbuf, "%s\\simparms.dat", startDir);
fp = fopen(inbuf, "rb");
if(fp == NULL)
  {
    strcpy(inbuf, "SIMPARMS.DAT file missing.
Default simulation parameters assumed.");
    error();
    return;
  }
fseek(fp, 0, SEEK_SET);
fread(&sp, sizeof(SP), 1, fp);
fclose(fp);
```

```
return;
}


void simInit(void)
{
readSimParms();
return;
}

void windowstat()
{
  int i, denom;
  float Total_busy_time, Total_time;
  float Idle_time, Busy_time, Error_overhead;
  float Response_time, Slack_time, Latency;
  float Throughput, Load, Success;

   if(transmitted == 0) // do not collect stats until at least
// one msg has been transmitted
    {
      strcpy(inbuf, "Too early for stats.
Not even one msg has been transmitted.");
      error();
      return;
    }

  if (sample_count == 1)
   {
     fprintf(wst,"\n\t\t\tSIMULATION STATISTICS \n\n");
     fprintf(wst,"\n\tNumber of nodes = %d\t\t", total_nodes);
     fprintf(wst,"Number of messages = %d\n\n", total_msgs);
     fprintf(wst,"\t.............................");
     fprintf(wst,".............................\n");
   }
  fprintf(wst, "\n");
  fprintf(wst,"\t------------------------------");
  fprintf(wst,"------------------------------\n");
  fprintf(wst,"\tNetwork Statistics\t\t");
  fprintf(wst,"Sampling Point %d at %d ms\n", sample_count,
(tic/sp.networkSpeed) );
  fprintf(wst,"\t------------------------------");
  fprintf(wst,"------------------------------\n");
  fprintf(wst,"\n\tTotal number of messages transmitted    = %10d",
               transmitted);
  fprintf(wst,"\n\tTotal number of remote messages         = %10d",
```

```
                remote);
fprintf(wst,"\n\tTotal number of collisions              = %10d",
                collisions);
fprintf(wst,"\n\tTotal number of msgs losing arbitration = %10d",
                losers);
fprintf(wst,"\n\tTotal number of errors encountered      = %10d",
                errors);
fprintf(wst,"\n\tTotal number of overload errors         = %10d",
                overload_count);
fprintf(wst,"\n\tTotal number of acknowledgement errors  = %10d",
                ack_count);
fprintf(wst,"\n\tTotal number of form errors             = %10d",
                form_count);
fprintf(wst,"\n\tTotal number of crc errors              = %10d",
                crc_count);
fprintf(wst,"\n\tTotal number of msgs resent             = %10d",
                retransmitted);

Idle_time = (float) idle_time / sp.networkSpeed ;
fprintf(wst,"\n\tIdle time in the network         = %10.2f ms",
                Idle_time);
Busy_time = (float) busy_time / sp.networkSpeed;
fprintf(wst,"\n\tBusy time in the network         = %10.2f ms",
                Busy_time);
Error_overhead = (float) error_overhead / sp.networkSpeed;
fprintf(wst,"\n\tError overhead time              = %10.2f ms",
                Error_overhead);

Response_time = (float) response_time / sp.networkSpeed /
                (float) transmitted;
fprintf(wst,"\n\tAverage response time            = %10.2f ms",
                Response_time);
Slack_time = (float) slack_time / sp.networkSpeed /
                (float) transmitted;
fprintf(wst,"\n\tAverage slack time               = %10.2f ms",
                Slack_time);
Latency = (float) latency / sp.networkSpeed /
                (float) transmitted;
fprintf(wst,"\n\tAverage latency time       = %10.2f ms",
                Latency);
Total_busy_time = Busy_time + Error_overhead;
Total_time = Busy_time + Error_overhead + Idle_time;
fprintf(wst,"\n\tSimulation time            = %10.2f ms",
                Total_time);
Load = (Total_busy_time / Total_time) * 100.0;
fprintf(wst,"\n\tNetwork load               = %10.2f %%", Load);
```

```
  Throughput = ( (float) transmitted / Total_time) * 1000.0;
  fprintf(wst,"\n\tNetwork throughput         = %10.2f msgs/s",
               Throughput);
  fprintf(g1,"%.2f %.2f\n", Total_time, Load);
  fprintf(g2,"%.2f %.2f\n", Load, Throughput);
  fprintf(g3,"%.2f %.2f\n", Total_time, Response_time);
  fprintf(g4,"%.2f %d\n", Load, collisions);
  fprintf(g5,"%.2f %.2f\n", Total_time, Latency);
  fprintf(g6,"%.2f %d\n", Load, errors);
  fprintf(g7, "%.2f %.2f\n", Load, Busy_time);
  fprintf(g8, "%.2f %.2f\n", Load, Idle_time);
  fprintf(g9, "%.2f %.2f\n", Load, Error_overhead);

  fprintf(wst,"\n\t------------------------------");
  fprintf(wst,"------------------------------\n");

  fprintf(wst,"\n\t\t\tNode Statistics\n");
  fprintf(wst,"\t------------------------------");
  fprintf(wst,"---------------\n");
  fprintf(wst,"\t------------------------------");
  fprintf(wst,"---------------\n");
  fprintf(wst,"\tNode       No of  No of   No of      Percent  \n");
  fprintf(wst,"\t           msgs   arbits  deadlines  sucess   \n");
  fprintf(wst,"\t           sent   lost    missed     in trans.\n");
  fprintf(wst,"\t------------------------------");
  fprintf(wst,"---------------\n");
  for (i = 0;i < total_nodes;i++)
   {
    fprintf(wst,"\n\t%-10s",node[i].node_name);
    fprintf(wst,"%5d",node[i].trans_count);
    fprintf(wst,"%8d",node[i].lost_count);
    fprintf(wst,"%8d",node[i].dead_count);
    denom = node[i].trans_count + node[i].error_count;
    if (denom != 0)
     {
      Success = (float) node[i].trans_count / (float) denom;
      fprintf(wst,"%11.2f",Success);
     }
    else
      fprintf(wst,"        --");
   }
  fprintf(wst,"\n\t------------------------------");
  fprintf(wst,"---------------\n");
}

void windowResults()
```

```
{
if( (tic >= sp.t1*250) && (tic <= sp.t2*250) )
windowstat();
}
```

```
// funcs1.c: This file contains some of the GUI functions.
#include <io.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <direct.h>
#include <bios.h>
#include <malloc.h>
#include <stdlib.h>
#include <process.h>

#include <qwadv.h>
#include <qwkeys.h>
#include "const.h"
#include "protos.h"
#include "defs.h"
void ltrim(char *);

int getFileName(void)
{
 char *tscrn;
 char **hlist1;
 char **vlist1;
 char strinp1[] = "      ";
 char *accept1 =  "      ";
 int win1;
 int Dialnumber, Flag, Value;
 int i, pos, dirpos;

 memmove(strinp1, po.filemask, strlen(po.filemask));
 tscrn = getmem(4000);

 prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
 hlist1 = malloc(numfiles * 4);
 for(i = 0; i < numfiles; i++)
  hlist1[i] = fileslist + i * 13;

 prepDirNames("*.*", _A_SUBDIR);
 vlist1 = malloc(numdirs * 4);
 for(i = 0; i < numdirs; i++)
  vlist1[i] = dirslist + i * 13;

 win1 = 0;
 wopen(5, 3, 75, 22, 184, 0x17, "Select a File", tscrn, &win1);
 dialoginit(win1, 5, 1);
```

```
 wcolor(win1, 0x30);
 inputbox(1, 2, 1, "File mask", 0x17, 416, 0, accept1, strinp1,
         0, 0);
 hlistbox(2, 1, 3, 15, 10, 3, 1, numfiles, hlist1, 0x17,
0x17, 0x17, 3);
 pushbutton(3, 7, 15, 8, 0x17, 0, "   Ok   ", 0x17);
 pushbutton(4, 31, 15, 8, 0x17, 0, " Cancel ", 0x17);
 vlistbox(5, 50, 3, 15, 10, 1, numdirs, vlist1, 0x17,
        0x17, 0x17, 3);
 pos = 1;
 dialogset(1);
 while(1)
  {
   Flag = 0;
   printa(1, 25, 0x30, spaceline);
   printa(1, 25, 0x30, curDir);
   printa(50, 25, 0x30, hlist1[pos-1]);
   while(Flag == 0 && Value != ESCAPE_KEY)
    dialogget(&Dialnumber, &Flag, &Value);
   if(Flag == 0 && Value == ESCAPE_KEY)
    {
     pos = 0;
     break;
    }
   if(Dialnumber == 2)
    pos = Value;
   if(Dialnumber == 4)
    {
     pos = 0;
     break;
    }
   if(Dialnumber == 1)
    {
     pos = 1;
     trim(strinp1);
     prepFileNames(strinp1, _A_NORMAL | _A_ARCH);
     hlist1 = realloc(hlist1, numfiles * 4);
     for(i = 0; i < numfiles; i++)
      hlist1[i] = fileslist + i * 13;
     hlistbox(2, 1, 3, 15, 10, 3, 1, numfiles, hlist1,
         0x17, 0x17, 0x17, 3);
     dlgupdate(2, 5, 0);
     if(noff)
      dialogset(1);
    }
   if(Dialnumber == 3)
```

```
      break;
    if(Dialnumber == 5)
     {
      dirpos = Value;
      chdir(vlist1[dirpos-1]);
      getcwd(curDir, 128);
      pos = 1;
      dirpos = 1;
      prepFileNames(strinp1, _A_NORMAL|_A_ARCH);
      hlist1 = realloc(hlist1, numfiles * 4);
      for(i = 0; i < numfiles; i++)
       hlist1[i] = fileslist + i * 13;

      prepDirNames("*.*", _A_SUBDIR);
      vlist1 = realloc(vlist1, numdirs * 4);
      for(i = 0; i < numdirs; i++)
       vlist1[i] = dirslist + i * 13;
      hlistbox(2, 1, 3, 15, 10, 3, 1, numfiles, hlist1,
          0x17, 0x17, 0x17, 3);
      vlistbox(5, 50, 3, 15, 10, 1, numdirs, vlist1,
         0x17, 0x17, 0x17, 3);
      dlgupdate(1, 5, 0);
      dlgupdate(5, 5, 0);
      continue;
     }
   }
  dlgclose();
  wclose(win1);
  free(tscrn);
  free(hlist1);
  free(vlist1);
  if(pos > 0 && !noff)
   strcpy(cfilename, fileslist + (pos-1) * 13);
  else
   return 0;
  return 1;
}

// return 1 if user has input a string of 1 or more chars;
// returns 0 if aborted

int getString(char *prompt, char *string, int lg, int spaceout)
{
 char *fgscrn;
 char *accept1 = "";
 int i, j, win1;
```

```
int Dialnumber, Flag, Value;

if(!spaceout)
 {
  i = strlen(string);
  if(i > lg)
   {
    strcpy(inbuf, "String not initialized properly");
    error();
    return 0;
   }
  while(i < lg)
  {
   string[i] = SPACE;
   i++;
  }
 }
else
 memset(string, SPACE, lg);
string[lg] = EOS;

fgscrn = getmem(1200);
win1 = 0;
wopen(10, 9, 70, 16, 184, 0x17, "", fgscrn, &win1);

dialoginit(win1, 3, 1);

wcolor(win1, 0x30);
inputbox(1, 2, 1, prompt, 0x17, 0, 0, accept1, string, 0, 0);
pushbutton(2, 15, 3, 8, 0x17, 0, "   Ok   ", 0x17);
pushbutton(3, 35, 3, 8, 0x17, 0, " Cancel ", 0x17);
Flag = 0;
while (Flag == 0 && Value != ESCAPE_KEY)
 dialogget(&Dialnumber, &Flag, &Value);

dlgclose();
wclose(win1);
free(fgscrn);

if(Flag == 0 && Value == ESCAPE_KEY)
 return 0;
if(Dialnumber == 3)
 return 0;
trim(string);
if(strlen(string))
 return 1;
```

```
 else
  return 0;
}

// 0 - Cancelled or invalid filename
// 1 - File does not exist or if it exists, it can be overwritten
// 2 - Filename is a sub-directory

int getFileNameMode1(void)
{
 char strinp1[41];
 int i, j, err = 0;

 i = getString("Filename ", strinp1, 40, 1);
 if(i)
  strcpy(cfilename, strinp1);
 else
  return 0;
 i = 0;
 i = strcspn(cfilename, " *?");
 if(i < strlen(cfilename))
  {
   strcpy(inbuf, "Invalid file name");
   error();
   return 0;
  }

 i = fexist(cfilename);
 if(i == 0)
  return 1;
 if(i == 1)
  {
   sprintf(inbuf, "File %s already exists. Overwrite?", cfilename);
   j = messagebox2(" No ", " Yes ", DBL_BOX | TRANS_SHADOW, 0x17,
         inbuf, 1, 50, 0x17, mbscrn);
   if(j == 1)
    return 0;
   if(j == 2)
    return 1;
  }
 if(i == 2)
  return 2;
}


int getFileNameMode2(void)
```

```
{
 char strinp1[41];
 int i, j;

 i = getString("Filename ", strinp1, 40, 1);
 if(i)
  strcpy(cfilename, strinp1);
 else
  return 0;

 i = fexist(cfilename);
 if(i == 0)
  return 1;
 if(i == 1)  //file
  {
   sprintf(inbuf, "File %s already exists", cfilename);
   j = messagebox3(" Cancel ", " Append ", " Overwrite ",
         DBL_BOX|TRANS_SHADOW,
        0x17, inbuf, 1, 40, 0x17, mbscrn);
   return (j-1);
  }
 if(i == 2)  //sub-dir
  {
   sprintf(inbuf, "%s is sub-directory", cfilename);
   error();
   return 0;
  }
}

int prepFileNames(char *mask, int fattr)  //32 normal files
{
 struct find_t find;

 numfiles = 0;
 if( !_dos_findfirst( mask, 0xffff, &find ) )
   if( find.attrib == fattr)  // only archived files
    ++numfiles;

  while( !_dos_findnext( &find ) )
   if( find.attrib == fattr)
    ++numfiles;

 if(!numfiles)
  {
   free(fileslist);
   fileslist = getmem(1 * 13);
```

```
    strcpy(fileslist, "No_files!");
    numfiles = 1;
    noff = 1; //set no files flag
    return 0;
   }
  else
   noff = 0; //clear no files flag

  fileslist = realloc(fileslist, numfiles * 13);

  if( !_dos_findfirst( mask, 0xffff, &find ) )
   if( find.attrib == fattr)
    {
     memmove(fileslist, find.name, 13);
     fileslist += 13;
    }
  while( !_dos_findnext( &find ) )
   if( find.attrib == fattr)
    {
     memmove(fileslist, find.name, 13);
     fileslist += 13;
    }

  fileslist -= numfiles * 13;
  qsort(fileslist, numfiles, 13, sortCompare);
  return numfiles;
}

int prepDirNames(char *mask, int fattr)
{
 struct find_t find;

 numdirs = 0;

 if( !_dos_findfirst( mask, 0xffff, &find ) )
   if( find.attrib == fattr)  // only archived files
    ++numdirs;

  while( !_dos_findnext( &find ) )
   if( find.attrib == fattr)
    ++numdirs;

 if(!numdirs)
  {
   dirslist = getmem(1 * 13);
   strcpy(dirslist, "\\");
```

```
  numdirs = 1;
  return 0;
 }

dirslist = realloc(dirslist, numdirs * 13);

if( !_dos_findfirst( mask, 0xffff, &find ) )
 if( find.attrib == fattr)
  {
   memmove(dirslist, find.name, 13);
   dirslist += 13;
  }
while( !_dos_findnext( &find ) )
 if( find.attrib == fattr)
  {
   memmove(dirslist, find.name, 13);
   dirslist += 13;
  }

dirslist -= numdirs * 13;
qsort(dirslist, numdirs, 13, sortCompare);
return numdirs;
}


int sortCompare(char *str1, char *str2)
{
 return strcmp(str1, str2);
}


void search(void)
{
 char *tscrn;
 char strinp1[] = "
 char *accept1 = "";
 char strinp2[] = "
 char *accept2 = "";
 char **hlist1;
 long start = 0, found;
 int i, j, numfound = 0;
 int win1;
 int Dialnumber, Flag, Value;
 int pos;

 tscrn = getmem(4000);
```

```
memmove(strinp1, srchstr, strlen(srchstr));
memmove(strinp2, po.filemask, strlen(po.filemask));
qwsystem(18, 1);
win1 = 0;
wopen(10, 7, 70, 16, 184, 0x17, "String search", tscrn, &win1);
dialoginit(win1, 4, 1);

wcolor(win1, 0x30);
inputbox(1, 2, 1, "String   ", 0x17, 0, 0, accept1, strinp1, 0, 0);
inputbox(2, 2, 3, "File mask", 0x17, 0, 0, accept2, strinp2, 0, 0);
wcolor(win1, 0x17);
pushbutton(3, 15, 5, 8, 0x17, 0, "   Ok   ", 0x17);
pushbutton(4, 35, 5, 8, 0x17, 0, " Cancel ", 0x17);
Flag = 0;
while (Flag == 0 && Value != ESCAPE_KEY)
 dialogget(&Dialnumber, &Flag, &Value);
dlgclose();
wclose(win1);

trim(strinp1);
trim(strinp2);
strcpy(srchstr, strinp1);
strcpy(srchmask, strinp2);
strcat(srchstr, " "); // to correct for an internal error in fscan

if(Flag == 0 && Value == ESCAPE_KEY)
 return;
if(Dialnumber == 4)
 return;

prepFileNames(srchmask, _A_NORMAL|_A_ARCH);
if(noff)
 {
  sprintf(inbuf, "No matching files found");
  error();
  return;
 }
hlist1 = malloc(numfiles * 4);
numfound = 0;
for(j = 0; j < numfiles; j++)
 {
  fscan(fileslist + j * 13, &start, srchstr, &found);
  if(found >= 0)
   {
    hlist1[numfound] = fileslist + j * 13;
    numfound++;
```

```
   }
  found = -1;
  start = 0;
 }

if(numfound)
 {
  sprintf(inbuf, "~cString  %s  was found in %d out of %d files",
                  strinp1, numfound, numfiles);
  win1 = 0;
  wopen(10, 4, 70, 18, 184, 0x17, "String search results", tscrn,
                  &win1);
  dialoginit(win1, 4, 1);
  hlistbox(1, 6, 1, 15, 4, 3, 1, numfound, hlist1, 0x17,
                  0x17, 0x17, 3);
  pushbutton(2, 5, 8, 8, 0x17, 0, "  Edit  ", 0x17);
  pushbutton(3, 25, 8, 8, 0x17, 0, "  View  ", 0x17);
  pushbutton(4, 45, 8, 8, 0x17, 0, " Cancel ", 0x17);
  wprints(win1, 2, 12, 0x17, inbuf);
  pos = 1;
  while(1)
   {
    Flag = 0;
    while(Flag == 0 && Value != ESCAPE_KEY)
     dialogget(&Dialnumber, &Flag, &Value);
    if(Flag == 0 && Value == ESCAPE_KEY)
     break;

    if(Dialnumber == 4)
     break;
    if(Dialnumber == 2)
     {
      getscrn(1,1,80,25,savearray);
      cson;
      bios_setcurtype(13, 14);
      sprintf(comm, "%s %s", po.editor, hlist1[pos-1]);
      system(comm);
      csoff;
      putscrn(1,1,80,25,savearray);
     }
    if(Dialnumber == 1)
     pos = Value;
    if(Dialnumber == 3)
     {
      getscrn(1,1,80,25,savearray);
      cson;
```

```
        bios_setcurtype(13, 14);
        sprintf(comm, "%s %s", po.browser, hlist1[pos-1]);
        system(comm);
        csoff;
        putscrn(1,1,80,25,savearray);
       }
     }
    dlgclose();
    wclose(win1);
    free(hlist1);
   }
  else
   {
    sprintf(inbuf, "String %s was not found in any of %d files",
          strinp1, numfiles);
    error();
   }

  free(tscrn);
  qwsystem(18, 0);
  return;
}

void printFile(void)
{
  unsigned int j;
  int prnport;
  char *buf;

  prnport = po.prnport - 1;

  if(!setPrinter())
   return;
  else
   {
    if(!getFileName())
     return;
    sprintf(inbuf, "Print file %s?", strupr(cfilename));
    j = messagebox2(" Yes ", " No ", DBL_BOX | TRANS_SHADOW,
          0x17, inbuf, 1, 50, 0x17, mbscrn);
    if(j == 2)
     return;
   }

  fp = fopen(cfilename, "rb");
  if(fp == NULL)
```

```
 return;
size = (unsigned long)  filelength(fileno(fp));
if(size <= 0)
 {
  sprintf(inbuf, "File %s is of zero length", cfilename);
  error();
  return;
 }

buf = getmem(size+1);
fread(buf, size, 1, fp);
fclose(fp);

for(j = 0; j < size; j++)
 _bios_printer( _PRINTER_WRITE, prnport, buf[j] );

_bios_printer( _PRINTER_WRITE, prnport, 12 ); //formfeed
free(buf);
return;
}
// This funtion sets the printer.
int setPrinter(void)
{
 int status;

 if(po.prnport < 1 || po.prnport > 3)
  {
   sprintf(inbuf, "Error: Invalid printer port.
Must be between 1 - 3");
   error();
   return 0;
  }
 status = lpstatus(po.prnport);
 if(status)
  {
   switch(status)
    {
     case PRN_READY:
      strcpy(inbuf, "Printer Ready");
      break;
     case PRN_OFFLINE:
      strcpy(inbuf, "Error: Printer is offline");
      break;
     case PRN_OFF:
      strcpy(inbuf, "Error: Printer is not on");
      break;
```

```
      case PRN_OFFLINE_NOPAPER:
       strcpy(inbuf, "Error: Printer is offline and has no paper");
       break;
      case PRN_ONLINE_NOPAPER:
       strcpy(inbuf, "Error: Printer has no paper");
       break;
      default:
       strcpy(inbuf, "Error: Printer is not connected.");
       break;
     }
    error();
    return 0;
   }
  else
   {
    strcpy(inbuf, "Printer Ok");
    return 1;
   }
}

// This functions does some file utilities.
void fileutil(void)
{
 char **hlist1;
 char **hlist2;
 char *tscrn, *buf;
 int i, j, pos, dirpos, win1;
 int Dialnumber, Flag, Value;
 char newname[26];
 int oldpos;

 prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
 hlist1 = malloc(numfiles * 4);
 for(i = 0; i < numfiles; i++)
  hlist1[i] = fileslist + i * 13;

 prepDirNames("*.*", _A_SUBDIR);
 hlist2 = malloc(numdirs * 4);
 for(i = 0; i < numdirs; i++)
  hlist2[i] = dirslist + i * 13;

 tscrn = getmem(4000);
 win1 = 0;
 wopen(9, 2, 72, 23, 184, 0x17, "File Utilities", tscrn, &win1);

 dialoginit(win1, 10, 1);
```

```
hlistbox(1, 2, 1, 15, 10, 2, 1, numfiles, hlist1,
        0x17, 0x17, 0x17, 3);
hlistbox(2, 42, 1, 15, 10, 1, 1, numdirs, hlist2,
        0x17, 0x17, 0x17, 3);
pushbutton(3,  3, 14, 8, 0x17, 0, " Cancel ", 0x17);
pushbutton(4, 18, 14, 8, 0x17, 0, "  View  ", 0x17);
pushbutton(5, 33, 14, 8, 0x17, 0, "  Copy  ", 0x17);
pushbutton(6, 48, 14, 8, 0x17, 0, " Merge  ", 0x17);
pushbutton(7,  3, 17, 8, 0x17, 0, " Delete ", 0x17);
pushbutton(8, 18, 17, 8, 0x17, 0, "  Move  ", 0x17);
pushbutton(9, 33, 17, 8, 0x17, 0, " Rename ", 0x17);
pushbutton(10, 48, 17, 8, 0x17, 0, "  Info  ", 0x17);
pos = 1;
dirpos = 1;
while(1)
 {
  Flag = 0;
  printa(1, 25, 0x30, spaceline);
  printa(1, 25, 0x30, curDir);
  printa(50, 25, 0x30, hlist1[pos-1]);

  while(Flag == 0 && Value != ESCAPE_KEY)
   dialogget(&Dialnumber, &Flag, &Value);
  if(Flag == 0 && Value == ESCAPE_KEY)
   break;

  if(Dialnumber == 1)
   pos = Value;
  if(Dialnumber == 3)
   break;
  if(Dialnumber == 2)
   {
    dirpos = Value;
    printa(1, 25, 0x30, spaceline);
    printa(1, 25, 0x30, curDir);
    chdir(hlist2[dirpos-1]);
    getcwd(curDir, 128);
    pos = 1;
    dirpos = 1;
      prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
    hlist1 = realloc(hlist1, numfiles * 4);
    for(i = 0; i < numfiles; i++)
     hlist1[i] = fileslist + i * 13;

    prepDirNames("*.*", _A_SUBDIR);
```

```
      hlist2 = realloc(hlist2, numdirs * 4);
      for(i = 0; i < numdirs; i++)
       hlist2[i] = dirslist + i * 13;
      hlistbox(1, 2, 1, 15, 10, 2, 1, numfiles, hlist1,
                       0x17, 0x17, 0x17, 3);
      hlistbox(2, 42, 1, 15, 10, 1, 1, numdirs, hlist2,
                       0x17, 0x17, 0x17, 3);
      dlgupdate(1, 5, 0);
      dlgupdate(2, 5, 0);
      continue;
     }
    if(noff)
     continue;
    if(Dialnumber == 4)
     {
      sprintf(comm, "%s %s", po.browser, hlist1[pos-1]);
      system(comm);
     }
    if(Dialnumber == 5)
     {
      dlgsave(savearray);
      j = getFileNameMode1();
      dlgrestore(savearray);
      if(j == 2)
       {
        strcpy(inbuf, cfilename);
        strcat(inbuf, "\\");
        strcat(inbuf, hlist1[pos-1]);
        i = fexist(inbuf);
        if(i == 2)
         {
          sprintf(inbuf, "Cannot copy %s to %s", hlist1[pos-1],
cfilename);
          j = 0;
         }
        if(i == 1)
         {
          sprintf(inbuf, "File %s exists. Overwrite?", inbuf);
          j = messagebox2(" No ", " Yes ", DBL_BOX | TRANS_SHADOW,
          0x17, inbuf, 1, 50, 0x17, mbscrn);
          j--;
         }
       }
      if(j > 0) //if not cancelled
       {
        fcopy(hlist1[pos-1], cfilename);
```

```
    prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
    hlist1 = realloc(hlist1, numfiles * 4);
    for(i = 0; i < numfiles; i++)
     hlist1[i] = fileslist + i * 13;
    hlistbox(1, 2, 1, 15, 10, 2, 1, numfiles, hlist1,
     0x17, 0x17, 0x17, 3);
    dlgupdate(1, 5, 0);
   }
 }
if(Dialnumber == 9)
 {
  dlgsave(savearray);
  sprintf(inbuf, "Rename %s to ", hlist1[pos-1]);
  j = getString(inbuf, newname, 12, 1);
  dlgrestore(savearray);
  if(j)
   {
    j = rename(hlist1[pos-1], newname);
    if(j)
     {
      sprintf(inbuf, "Invalid or duplicate file name");
      error();
    }
    else
     {
      strupr(newname);
      strcpy(fileslist + (pos-1) * 13, newname);
      qsort(fileslist, numfiles, 13, sortCompare);
      for(i = 0; i < numfiles; i++)
       hlist1[i] = fileslist + i * 13;
      dlgupdate(1, 5, 0);
     }
   }
 }
if(Dialnumber == 8)
 {
  dlgsave(savearray);
  sprintf(inbuf, "Move %s to directory", hlist1[pos-1]);
  j = getString(inbuf, newname, 25, 1);
  dlgrestore(savearray);
  if(j)
   {
    j = fmove(hlist1[pos-1], newname);
    if(j)
     {
      sprintf(inbuf, "Error: Invalid directory name.");
```

```
        error();
       }
      else
       {
prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
       hlist1 = realloc(hlist1, numfiles * 4);
       for(i = 0; i < numfiles; i++)
        hlist1[i] = fileslist + i * 13;
       hlistbox(1, 2, 1, 15, 10, 2, 1, numfiles, hlist1,
       0x17, 0x17, 0x17, 3);
       dlgupdate(1, 5, 0);
      }
    }
   }
  if(Dialnumber == 6)
   {
    dlgsave(savearray);
    sprintf(inbuf, "Attach %s to", hlist1[pos-1]);
    j = getString(inbuf, newname, 12, 1);
    dlgrestore(savearray);
    if(j)
     {
      sprintf(inbuf,"%s will be appended to %s", hlist1[pos-1],
                    strupr(newname));
      j = messagebox2(" Ok ", " Cancel ", DBL_BOX | TRANS_SHADOW,
       0x17, inbuf, 1, 50, 0x17, mbscrn);
      if(j == 1)
       {
        j = fappend(newname, hlist1[pos-1]);
        if(j)
         {
          sprintf(inbuf, "Error: Merge was not successful");
          error();
         }
       }
     }
   }
  if(Dialnumber == 10)
   {
    if(kbhit())
     getch();
    showfileinfo(hlist1[pos-1]);
   }
  if(Dialnumber == 7)
   {
    sprintf(inbuf, "Delete file %s ?", hlist1[pos-1]);
```

```
      j = messagebox2(" No ", " Yes ", DBL_BOX | TRANS_SHADOW,
                                  0x17, inbuf, i, 50, 0x17, mbscrn);
      if(j == 2)
       fkill(hlist1[pos-1]);
      else
       continue;
      prepFileNames(po.filemask, _A_NORMAL|_A_ARCH);
      for(i = 0; i < numfiles; i++)
       hlist1[i] = fileslist + i * 13;
      hlistbox(1, 2, 1, 15, 10, 2, 1, numfiles, hlist1,
          0x17, 0x17, 0x17, 3);
      dlgupdate(1, 5, 0);
      pos = 1;
     }
    if(Dialnumber == 2)
     dialogset(2);
    else
     dialogset(1);
   }
 dlgclose();
 wclose(win1);
 free(hlist1);
 free(hlist2);
 free(tscrn);
 return;
}

void setFilemask(void)
{
 int j;
 char strinp1[13];

 strcpy(strinp1, po.filemask);
 j = getString("File mask ", strinp1, 12, 0);
 if(j)
   strcpy(po.filemask, strinp1);
 return;
}

void sysinfo()
{
 int card, monitor;
 char *tscrn;
 unsigned i, j, k;
 int wid;
 struct diskfree_t drvinfo;
```

```
unsigned drive, drivecount, memory, pstatus;
union
 {                              // Access equiment either as:
  unsigned u;                   // unsigned or
  struct                        // bit fields
  {
  unsigned diskflag : 1;        // Diskette drive installed?
  unsigned coprocessor : 1;     // Coprocessor? (except on PC)
  unsigned sysram : 2;          // RAM on system board
  unsigned video : 2;           // Startup video mode
  unsigned disks : 2;           // Drives 00=1, 01=2, 10=3, 11=4
  unsigned dma : 1;             // 0=Yes, 1=No (1 for PC Jr.)
  unsigned comports : 3;        // Serial ports
  unsigned game : 1;            // Game adapter installed?
  unsigned modem : 1;           // Internal modem?
  unsigned printers : 2;        // Number of printers
  } bits;
} equip;
char y[] = "YES", n[] = "NO";
char *vidcard[] = {"None", "MDA", "CGA", "EGA",
"MCGA", "VGA", "Hercules"};
char *mon[] = {"None", "Monochrome", "Color", "Color",
"Monochrome", "Color"};

tscrn = getmem(4000);
wid = 0;
wopen(5, 3, 77, 23, SNG_BOX | TRANS_SHADOW, 0x17, "", tscrn, &wid);
wtitle(wid, 0, 0x17, "System Information");
_dos_getdrive( &drive );
wprintf(wid, "\nCurrent drive:\t\t\t%c:\n", 'A' + drive - 1 );

_dos_getdiskfree( drive, &drvinfo );
wprintf(wid, "Disk space free:\t\t%ld\n",
(long)drvinfo.avail_clusters *
drvinfo.sectors_per_cluster *
drvinfo.bytes_per_sector );

// Get new drive and number of logical drives in system.
_dos_getdrive( &drive );
wprintf(wid, "Number of logical drives:\t%d\n", numdrives );

memory = _bios_memsize();
wprintf(wid, "Memory:\t\t\t\t%dK\n", memory );

equip.u = _bios_equiplist();
```

```
wprintf(wid, "Disk drive installed:\t\t%s\n",
              equip.bits.diskflag ? y : n );
wprintf(wid, "Coprocessor installed:\t\t%s\n",
              equip.bits.coprocessor ? y : n );
wprintf(wid, "Game adapter installed:\t\t%s\n",
              equip.bits.game ? y : n );
wprintf(wid, "Serial ports:\t\t\t%d\n",
              equip.bits.comports );
wprintf(wid, "Number of printers:\t\t%d\n",
              equip.bits.printers );

// Fail if any error bit is on, or if either operation bit is off.
for(i = 0; i < equip.bits.printers; i++)
 {
  pstatus = _bios_printer( _PRINTER_STATUS, i, 0 );
  if( (pstatus & 0x29) || !(pstatus & 0x80) || !(pstatus & 0x10) )
   pstatus = 0;
  else
   pstatus = 1;
   wprintf(wid, "LPT%i Printer available:\t\t%s\n",
i+1, pstatus ? y : n );
 }
vidtype(&card, &monitor, &i, &j, &k);
wprintf(wid,"\nVideo card:\t\t\t%s", vidcard[card]);
wprintf(wid,"\nMonitor:\t\t\t%s", mon[monitor]);
wprintf(wid,"\nPress any key to continue...");
waitevent();
wclose(wid);
free(tscrn);
return;
}

void setDirDrive(void)
{
 char *tscrn;
 char **hlist1;
 char **vlist1;
 int win1;
 int Dialnumber, Flag, Value;
 int i, drvpos;
 int j, olddrive;

 tscrn = getmem(4000);

 prepDirNames("*.*", _A_SUBDIR);
 hlist1 = malloc(numdirs * 4);
```

```
for(i = 0; i < numdirs; i++)
 hlist1[i] = dirslist + i * 13;

vlist1 = malloc(numdrives * 4);
for(i = 0; i < numdrives; i++)
 vlist1[i] = driveslist + i * 6;


win1 = 0;
wopen(5, 5, 75, 22, 184, 0x17, "Set Drive and Directory",
              tscrn, &win1);
dialoginit(win1, 3, 1);
wcolor(win1, 0x30);
hlistbox(1, 1, 1, 15, 10, 3, 1, numdirs, hlist1,
              0x17, 0x17, 0x17, 3);
pushbutton(2, 30, 13, 8, 0x17, 0, "   Ok   ", 0x17);
vlistbox(3, 50, 1, 15, 10, 1, numdrives, vlist1,
              0x17, 0x17, 0x17, 3);
while(1)
 {
  Flag = 0;
  printa(1, 25, 0x30, spaceline);
  printa(1, 25, 0x30, curDir);

  while(Flag == 0 && Value != ESCAPE_KEY)
   dialogget(&Dialnumber, &Flag, &Value);
  if(Flag == 0 && Value == ESCAPE_KEY)
   break;

  if(Dialnumber == 3)
   {
    drvpos = *(driveslist + (Value - 1) * 6 + 2) - 64;
    olddrive = _getdrive();
    j = drvstatus(drvpos);
    if(j)
     {
      sprintf(inbuf, "General failure reading drive %c:", drvpos+64);
      if(j == 15)
       sprintf(inbuf, "%c: is an invalid drive", drvpos+64);
      if(j == 21)
       sprintf(inbuf, "%c: Drive not ready", drvpos+64);
      error();
      _chdrive(olddrive);
      dialogset(3);
      continue;
     }
```

```
    _chdrive(drvpos);
    getcwd(curDir, 128);
    prepDirNames("*.*", _A_SUBDIR);
    hlist1 = realloc(hlist1, numdirs * 4);
    for(i = 0; i < numdirs; i++)
     hlist1[i] = dirslist + i * 13;
    hlistbox(1, 1, 1, 15, 10, 3, 1, numdirs, hlist1,
                0x17, 0x17, 0x17, 3);
    dlgupdate(1, 5, 0);
    Value = 1;
   }
  if(Dialnumber == 2)
    break;
  if(Dialnumber == 1)
   {
    chdir(hlist1[Value-1]);
    getcwd(curDir, 128);
    prepDirNames("*.*", _A_SUBDIR);
    hlist1 = realloc(hlist1, numdirs * 4);
    for(i = 0; i < numdirs; i++)
     hlist1[i] = dirslist + i * 13;
    hlistbox(1, 1, 1, 15, 10, 3, 1, numdirs, hlist1,
                0x17, 0x17, 0x17, 3);
    dlgupdate(1, 5, 0);
    Value = 1;
    dialogset(1);
   }
 }
dlgclose();
wclose(win1);
free(tscrn);
free(hlist1);
free(vlist1);
return;
}


void shell(void)
{
 mhide();
 getscrn(1,1,80,25,savearray);
 bios_cls(0x07);
 printa(1,1,0x07,"Type 'EXIT' to return to CANSIM");
 cson;
 bios_setcurtype(13, 14);
 system("COMMAND.COM");
```

```
 csoff;
 putscrn(1,1,80,25,savearray);
 mshow();
 return;
}

void exitmm(void)
{
 free(savearray);
 free(mbscrn);
 free(fileslist);
 free(subslist);
 free(dirslist);
 free(driveslist);
 chdir(startDir);
 bios_cls(0x07);
 qwexit(CURSOR);
 exit(0);
}

void error(void)
{
 qwbeep();
 messagebox(" Ok ", DBL_BOX | TRANS_SHADOW, 0x67, inbuf, 1,
              strlen(inbuf)+4, 0x17, mbscrn);
 return;
}

void msg(void)
{
 messagebox(" Ok ", DBL_BOX | TRANS_SHADOW, 0x67, inbuf, 1,
              strlen(inbuf)+4, 0x17, mbscrn);
 return;
}

void trim(char *string)
{
 int i;

 i = strlen(string) - 1;
 if(i < 0) return;
 while(string[i] == SPACE && i >= 0)
  i--;
 string[i+1] = EOS;
 return;
}
```

```
void ltrim(char *string)
{
 int i, lg;
 char *buf;

 lg = strlen(string);
 if(!lg)
  return;
 buf = malloc(lg+1);
 strcpy(buf, string);
 i = 0;
 while(string[i] == SPACE)
  i++;
 strcpy(string, buf + i);
 free(buf);
 return;
}


char *getmem(int size)
{
 char *ptr;

 ptr = malloc(size);
 if(ptr != NULL)
  return ptr;
 qwbeep();
 strcpy(inbuf, "Memory allocation error");
 error();
 exit(-1);
 return NULL;
}
// This routine shows the information about a file.
void showfileinfo(char *fname)
{
 int i, win2;
 char *tscrn2;
 QWDIR qfile;

 memset(&qfile, 0, sizeof(QWDIR));
 i = 0;
 rddir(fname, 0xffff, &i, &qfile);
 tscrn2 = getmem(1500);
 win2 = 0;
 wopen(20, 8, 60, 18, 184, 0x17, "File information", tscrn2, &win2);
```

```
wprintf(win2, "\nFile name : %s", qfile.name);
wprintf(win2, "\nSize      : %-7lu bytes", qfile.size);
wprintf(win2, "\nDate      : %02u-", qfile.month);
wprintf(win2, "%02u-", qfile.day);
wprintf(win2, "%4u", qfile.year+1980);
wprintf(win2, "\nTime      : %02u:", qfile.hours);
wprintf(win2, "%02u:", qfile.mins);
wprintf(win2, "%02u", qfile.secs);
wprintf(win2, "\nAttributes: ");
wprintf(win2, qfile.ro ? "RDO ": "");
wprintf(win2, qfile.hid ? "HID ": "");
wprintf(win2, qfile.sys ? "SYS ": "");
wprintf(win2, qfile.dir ? "DIR ": "");
wprintf(win2, qfile.arc ? "ARC ": "");
wprintf(win2, "\n\nPrint any key to continue...");
waitevent();
wclose(win2);
free(tscrn2);
return;
}

void viewFile(char *fname)
{
 int i, j, k, barattr, num, rpos, flag, wid1;
 int kb, options;
 char *buf, *nbuf;
 char **lines;
 char *tscrn;

 if(noff)
  return;
 tscrn = getmem(4000);
 fp = fopen(fname, "rb");
 size = (unsigned long) filelength(fileno(fp));
 buf = getmem(size+1);
 fread(buf, size, 1, fp);
 fclose(fp);
 buf[size] = EOS;

 num = 50;
 lines = malloc(num * 4);
 strcpy(inbuf, fname);
 lines[0] = inbuf;
 lines[1] = buf;
```

```
i = 1;
nbuf = buf;
while(1)
 {
  if((nbuf = strchr(nbuf, CR)) != NULL)
   {
    *nbuf = EOS;
    i++;
    nbuf += 2;
    if(i+2 > num)
     {
      num += 50;
      lines = realloc(lines, num * 4);
     }
    lines[i] = nbuf;
   }
  else
   break;
 }
num = i;

wid1 = 0;
wopen(1, 2, 80, 24, 0, 0x17, "", tscrn, &wid1);
wwrap(wid1, 0);
barattr = 0x0f;
rpos = 1;
flag = 1;
options = HM_HASTITLE; //HM_USEWPRINTS |
wcsron(wid1);
while(1)
 {
  helpmenu(wid1, options, barattr, num, lines, &rpos, &kb, &flag);
  if ((flag == 2)&&(kb == ESCAPE_KEY))
   break;
  i = 0;
  mouse(&i, &j, &k);
  if(i > 1) //rt button or rt+lf buttons
   break;
  flag = 0;
 }
wclose(wid1);
free(buf);
free(lines);
free(tscrn);
return;
}
```

```
void backup(void)
{
 char *tscrn;
 char strinp1[] = "                ";
 char accept1[] = "";
 char strinp2[] = "                ";
 char accept2[] = "";
 char strinp3[] = "                ";
 char accept3[] = "";
 int win1, j;
 int Dialnumber, Flag, Value;

 tscrn = getmem(2000);
 win1 = 0;
 wopen(8, 7, 74, 19, 184, 0x17, "File Back-up", tscrn, &win1);
 wprinta(win1, 1, 1, 0x17, "File mask");
 wprinta(win1, 1, 2, 0x17, "Source directory");
 wprinta(win1, 1, 3, 0x17, "Back-up directory");
 qwsystem(18,1);
 dialoginit(win1, 5, 1);
 wcolor(win1, 0x30);
 memmove(strinp1, po.filemask, strlen(po.filemask));
 memmove(strinp2, curDir, strlen(curDir));
 memmove(strinp3, po.bakupDir, strlen(po.bakupDir));
 inputbox(1, 19, 1, "", 0x17, 0, 0, accept1, strinp1, 0, 0);
 inputbox(2, 19, 2, "", 0x17, 0, 0, accept2, strinp2, 0, 0);
 inputbox(3, 19, 3, "", 0x17, 0, 0, accept3, strinp3, 0, 0);
 wcolor(win1, 0x17);
 pushbutton(4, 18, 6, 8, 0x17, 0, " Start  ", 0x17);
 pushbutton(5, 40, 6, 8, 0x17, 0, " Cancel ", 0x17);
 while(1)
  {
   dialogget(&Dialnumber, &Flag, &Value);
   if(Flag)
    {
     if(Dialnumber == 5)
      break;
     if(Dialnumber == 4)
      {
       trim(strinp1);
       trim(strinp2);
       trim(strinp3);
       ltrim(strinp1);
```

```
        ltrim(strinp2);
        ltrim(strinp3);
        j = filebackup(strinp1, strinp2, strinp3);
        if(j == 0)
         break;
        else
         dialogset(1);
       }
     }
   }
 dlgclose();
 wclose(win1);
 free(tscrn);
 return;
}

int filebackup(char *mask, char *srcdir, char *dstdir)
// returns 0 = OK, exit; 1 continue dialog
{
 typedef struct
 {
  int  year;
  int  month;
  int  day;
  int  hours;
  int  mins;
  int  secs;
 } FI;
 char src[257], dst[257];
 QWDIR dqfile, sqfile;
 FI *sf;
 int flag, i, j, *copy, olddrive, srcdrv, dstdrv, numcopied = 0;
 int retVal = 1;

 sf = malloc(1 * sizeof(FI));
 copy = malloc(sizeof(int));
 strupr(srcdir);
 strupr(dstdir);
 if(strcmp(srcdir, dstdir) == 0)
  {
   strcpy(inbuf, "Source and back-up directories
cannot be the same");
   error();
   goto quitbakup;
  }
```

```
olddrive = _getdrive();
if(strchr(srcdir, ':'))
 {
  srcdrv = (int) *srcdir - 64;
  if(strlen(srcdir) < 2)
   goto quitbakup;
  if(strlen(srcdir) == 2)
   _getdcwd(srcdrv, srcdir, 40);
 }
else
 srcdrv = olddrive;

j = drvstatus(srcdrv);
if(j)
 {
  sprintf(inbuf, "General failure reading drive %c:", srcdrv+64);
  if(j == 15)
   sprintf(inbuf, "%c: is an invalid drive", srcdrv+64);
  if(j == 21)
   sprintf(inbuf, "%c: Drive not ready", srcdrv+64);
  error();
  goto quitbakup;
 }
else
 _chdrive(srcdrv);

j = chdir(srcdir);
if(j != 0)
 {
  strcpy(inbuf, "Source directory does not exist");
  error();
  goto quitbakup;
 }
prepFileNames(mask, _A_ARCH|_A_NORMAL);
if(noff)
 {
  strcpy(inbuf, "Source directory does not contain
any matching files");
  error();
  goto quitbakup;
 }
sf = realloc(sf, numfiles * sizeof(FI));
memset(sf, 0, numfiles * sizeof(FI));
for(i = 0; i < numfiles; i++)
 {
  flag = 0;
```

```
  memset(&sqfile, 0, sizeof(QWDIR));
  rddir(fileslist + i * 13, 0xffff, &flag, &sqfile);
  sqfile.secs  = sf[i].secs;
  sqfile.mins  = sf[i].mins;
  sqfile.hours = sf[i].hours;
  sqfile.day   = sf[i].day;
  sqfile.month = sf[i].month;
  sqfile.year  = sf[i].year;
 }

if(strchr(dstdir, ':'))
 {
  dstdrv = (int) *dstdir - 64;
  if(strlen(dstdir) < 2)
   goto quitbakup;
  if(strlen(dstdir) == 2)
   _getdcwd(dstdrv, dstdir, 40);
 }
else
 dstdrv = olddrive;

j = drvstatus(dstdrv);
if(j)
 {
  sprintf(inbuf, "General failure reading drive %c:", dstdrv+64);
  if(j == 15)
   sprintf(inbuf, "%c: is an invalid drive", dstdrv+64);
  if(j == 21)
   sprintf(inbuf, "%c: Drive not ready", dstdrv+64);
  error();
  goto quitbakup;
 }
else
 _chdrive(dstdrv);

j = chdir(dstdir);
if(j != 0)
 {
  strcpy(inbuf, "Back-up directory does not exist");
  error();
  goto quitbakup;
 }
copy = realloc(copy, numfiles * sizeof(int));
memset(copy, 0, numfiles * sizeof(int));
for(i = 0; i < numfiles; i++)
 {
```

```
  j = fexist(fileslist + i * 13);
  if(j == 0)
   copy[i] = 1;
  if(j == 1)
   {
    memset(&dqfile, 0, sizeof(QWDIR));
    flag = 0;
    rddir(fileslist + i * 13, 0xffff, &flag, &dqfile);
    j = 0;
    while(1)
     {
      if(dqfile.year > sf[i].year)
       break;
      if(dqfile.year < sf[i].year)
       {j = 1; break;}
      if(dqfile.month > sf[i].month)
       break;
      if(dqfile.month < sf[i].month)
       {j = 1; break;}
      if(dqfile.day > sf[i].day)
       break;
      if(dqfile.day < sf[i].day)
       {j = 1; break;}
      if(dqfile.hours > sf[i].hours)
       break;
      if(dqfile.hours < sf[i].hours)
       {j = 1; break;}
      if(dqfile.mins  > sf[i].mins)
       break;
      if(dqfile.mins < sf[i].mins)
       {j = 1; break;}
      if(dqfile.secs  > sf[i].secs)
       break;
      if(dqfile.secs < sf[i].secs)
       {j = 1; break;}
      else
       break;
     }
    copy[i] = j;
   }
 }

j = 0;
for(i = 0; i < numfiles; i++)
 j += copy[i];
```

```
if(j == 0)
 {
  strcpy(inbuf, "All files up to date. No back-up performed");
  msg();
  retVal = 0;
  goto quitbakup;
 }
_chdrive(srcdrv);
chdir(srcdir);
wprintc(1, 9, 0x1e, "Backing-up files. Please wait...");
for(i = 0; i < numfiles; i++)
 {
  if(copy[i])
   {
    strcpy(src, fileslist + i * 13);
    wprintc(1, 10, 0x1e, src);
    j = fcopy(src, dstdir);
    if(j == 0)
     numcopied++;
    wprintc(1, 10, 0x1e, "            ");
   }
 }
wprintc(1, 9, 0x17, "                                        ");
sprintf(inbuf, "%d File(s) backed-up", numcopied);
msg();
retVal = 0;
quitbakup:
 _chdrive(olddrive);
 free(sf);
 free(copy);
 chdir(curDir);
 return retVal;
}
```

```
// funcs2.c: This file contains some of the GUI functions.

#include <bios.h>
#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include <search.h>
#include <direct.h>

#include <qwadv.h>
#include <qwkeys.h>
#include "const.h"
#include "protos.h"
#include "defs.h"


void setup(void)
{
PO temp;
char *tscrn;

char *strinp1 = "        ";
char *accept1 = "        ";
char *strinp2 = "        ";
char *accept2 = "        ";
char *strinp3 = "        ";
char *accept3 = "        ";
char *strinp4 = "        ";
char *accept4 = "        ";
char *strinp5 = "        ";
char *accept5 = "        ";
char *strinp6 = "        ";
char *accept6 = "123";

int win1;
int Dialnumber, Flag, Value;

kbsystem(3, DOWN_ARROW_KEY);
kbsystem(4, UP_ARROW_KEY);
qwsystem(18, 1);
qwsystem(12, 32);
tscrn = getmem(4000);
memmove(&temp, &po, sizeof(PO));     //hold po in temp

memmove(strinp1, po.editor, strlen(po.editor));
memmove(strinp2, po.browser, strlen(po.browser));
```

```
memmove(strinp3, po.dataFilesDir, strlen(po.dataFilesDir));
memmove(strinp5, po.filemask, strlen(po.filemask));
memmove(strinp4, po.bakupDir, strlen(po.bakupDir));
strinp6[0] = po.prnport + 48;
strinp6[1] = 0;

win1 = 0;
wopen(6, 6, 75, 18, 49, 0x17, "Program parameters", tscrn, &win1);
dialoginit(win1, 9, 1);
wcolor(win1, 0x71);
inputbox(1,5,1,"File editor", 0x17, 416, 0, accept1, strinp1, 0, 0);
inputbox(2,5,2,"File viewer", 0x17, 410, 0, accept2, strinp2, 0, 0);
inputbox(3,5,3,"Default data dir ", 0x17, 410, 0, accept3, strinp3,
0, 0);
inputbox(4,5,4,"Default bakup dir", 0x17, 410, 0, accept4, strinp4,
0, 0);
inputbox(5,5,5,"Default file extn", 0x17, 410, 0, accept5, strinp5,
0, 0);
inputbox(6,5,6,"Printer port     ", 0x17, 410, 0, accept6, strinp6,
0, 0);
pushbutton(7, 9, 8, 0, 0x17, 0, "   OK   ", 0x30);
pushbutton(8, 28, 8, 0, 0x17, 0, "   Save   ", 0x30);
pushbutton(9, 49, 8, 0, 0x17, 0, "  Cancel  ", 0x30);

while(1)
{
Flag = 0;
Value = ESCAPE_KEY - 1;
while(Flag == 0 && Value != ESCAPE_KEY)
dialogget(&Dialnumber, &Flag, &Value);
if(Flag == 0 && Value == ESCAPE_KEY)
break;
if(Dialnumber >= 7)
break;
}
trim(strinp1);
strcpy(po.editor, strinp1);
trim(strinp2);
strcpy(po.browser, strinp2);
trim(strinp3);
strcpy(po.dataFilesDir, strinp3);
trim(strinp5);
strcpy(po.filemask, strinp5);
trim(strinp4);
strcpy(po.bakupDir, strinp4);
po.prnport = atoi(strinp6);
```

```
if(Dialnumber == 8) //save
saveSettings();
if(Dialnumber == 9)  //
memmove(&po, &temp, sizeof(PO));
wclose(win1);
free(tscrn);
kbsystem(3, 0);
kbsystem(4, 0);
qwsystem(18, 0);
return;
}

int saveSettings(void)
{
sprintf(inbuf, "%s\\setup.dat", startDir);
fp = fopen(inbuf, "wb");
if(fp == NULL)
   return 0;
fwrite(&po, sizeof(PO), 1, fp);
fclose(fp);
return(1);
}

void readSettings(void)
{
int i;

strcpy(po.browser, "C:\\CANSIM\\BROWSE2.COM");
strcpy(po.dataFilesDir, "C:\\CANSIM");
strcpy(po.filemask, "*.*");
strcpy(po.bakupDir, "C:\\CANSIM\\BAK");
po.prnport = 1;
for(i = 0; i < 10; i++)
strcpy(po.masks[i], "*.*");

fp = fopen("setup.dat", "rb");
if(fp == NULL)
{
strcpy(inbuf, "SETUP.DAT file missing. Default parameters assumed.");
error();
return;
}
fread(&po, sizeof(PO), 1, fp);
fclose(fp);
return;
```

```
}


void welcome(void)
{
int i, j, k, card, monitor;
int win1;
char *fake;

vidtype(&card, &monitor, &i, &j, &k);
if(monitor == 1 || monitor == 4) //monochrome monitor
makemono(MMONO_NOBLINK);

if(card == MDA)
vidmode(0x07);    //80 x 25 color
else
vidmode(0x03);    //80 x 25 BW

fillscrn(1, 1, 80, 25, 0xb2, 0x17);

win1 = 0;
wopenn(10, 8, 70, 16, 50, 0x17, "", fake, &win1);
wprintc(win1, 0, 0x17,"W E L C O M E   T O   C A N
S I M U L A T O R!");
wprintc(win1, 3, 0x17, "Created by Mujibur Rehman");
wprintc(win1, 6, 0x17, "Press any key to continue...");
waitevent();
wclosen(win1);
fillscrn(1, 1, 80, 25, 0xb2, 0x17);
kbsystem(5, 0x0000);
kbsystem(9, 0x5200);

firsttime = 1;

savearray = getmem(6000);
mbscrn = getmem(4000);
fileslist = getmem(100);
subslist = malloc(10 * sizeof(PROG));
dirslist = getmem(100);
memset(spaceline, SPACE, 80);
getcwd(startDir, 128);
readSettings();
j = chdir(po.dataFilesDir);
if(j != 0)
{
```

```
strcpy(inbuf, "Data files directory not found");
error();
}
getcwd(curDir, 128);
printa(1, 25, 0x30, spaceline);
printa(1, 25, 0x30, curDir);
strcpy(srchstr, "  ");
strcpy(srchmask, po.filemask);

driveslist = getmem(26 * 6);
numdrives = 0;
for(i = 1; i <= 26; i++)
{
if(drvstatus(i) == 15)
continue;
sprintf(driveslist + numdrives * 6, "[-%c-]", i + 64);
numdrives++;
}
return;
}

void waitevent(void)
{
int i, j, k;
while(1)
{
i = kbhit();
if(i)
getch();
else
mouse(&i, &j, &k);
if(i)
break;
}
return;
}


void printStr(char *buf, int numlines)
{
int i, prnport;
prnport = po.prnport - 1;
for(i = 0; i < numlines; i++)
{
_bios_printer( _PRINTER_WRITE, prnport, CR );
_bios_printer( _PRINTER_WRITE, prnport, LF );
```

```
}
while(*buf != EOS)
{
_bios_printer( _PRINTER_WRITE, prnport, *buf);
buf++;
}
}


void popmasks(void)
{
int x, y, style, num, key, rpos, flag;
int battr, cattr, dattr;
char *scrn;
int i, abort = 0;
char *menu_items[15];

for(i = 0; i < 15; i++)
menu_items[i] = malloc(15);

strcpy(menu_items[0], "File masks");
for(i = 1; i <= 10; i++)  //10 was here
{
strcpy(menu_items[i], "  ");
strcpy((menu_items[i]) + 2, po.masks[i-1]);
}
strcpy(menu_items[i], "");
i++;
strcpy(menu_items[i], "  Other mask");
i++;
strcpy(menu_items[i], "  Define masks");
i++;
strcpy(menu_items[i], "  Cancel");

scrn = getmem(1000);
y = 2;
x = 35;
style = SNG_BOX | TRANS_SHADOW;
battr = 0x17;
cattr = 0x17;
dattr = 0x7b;
num  = 14;
rpos = 1;
flag = 1;

while(1)
```

```
{
  popmenu(x, y, style, battr, cattr, dattr, num,
menu_items, scrn, &rpos, &key, &flag);
if(!flag)
continue;
if(flag != 1)
abort = 1;
break;
}
popclose();
free(scrn);
if(!abort && rpos < 11)
strcpy(po.filemask, menu_items[rpos] + 2);
for(i = 0; i < 15; i++)
free(menu_items[i]);
if(!abort && rpos == 13)
defineMasks();
return;
}


void defineMasks(void)
{
int updFlag, saveFlag;
PO temp;
char *tscrn;
char *accept = "";
char prompt[15];
char strinp[10][13];
int i, win1;
int Dialnumber, Flag, Value;

kbsystem(3, DOWN_ARROW_KEY);
kbsystem(4, UP_ARROW_KEY);
qwsystem(18, 1);
qwsystem(12, 32);
tscrn = getmem(4000);
memmove(&temp, &po, sizeof(PO));

win1 = 0;
wopen(13, 5, 69, 22, 50, 0x17, "Define file masks", tscrn,
&win1);
wcolor(win1, 0x30);
dialoginit(win1, 13, 1);
for(i = 0; i < 10; i++)
{
```

```
strcpy(strinp[i], "            ");
memmove(strinp[i], po.masks[i], strlen(po.masks[i]));
sprintf(prompt, "File mask %2d ", i+1);
inputbox(i+1, 1, i+1, prompt, 0x17, 0, 0, accept, strinp[i],
0, 0);
}
wcolor(win1, 0x17);
wprinta(win1, 0, 12, 0x17, "-------------------------");
pushbutton(11,  5, 13, 1, 0x17, 0, "   Ok   ", 0x17);
pushbutton(12, 23, 13, 1, 0x17, 0, "  Save  ", 0x17);
pushbutton(13, 41, 13, 1, 0x17, 0, " Cancel ", 0x17);

saveFlag = 0;
updFlag = 0;

while(1)
{
Flag = 0;
Value = ESCAPE_KEY - 1;
while(Flag == 0 && Value != ESCAPE_KEY)
dialogget(&Dialnumber, &Flag, &Value);
if(Flag == 0 && Value == ESCAPE_KEY)
break;

if(Dialnumber == 13)
break;
if(Dialnumber == 12)
{
updFlag = 1;
saveFlag = 1;
break;
}
if(Dialnumber == 11)
{
updFlag = 1;
break;
}
}
for(i = 0; i < 10; i++)
{
trim(strinp[i]);
strcpy(po.masks[i], strinp[i]);
}

if(saveFlag)
saveSettings();
```

```
if(!updFlag)
memmove(&po, &temp, sizeof(PO));
wclose(win1);
free(tscrn);
kbsystem(3, 0);
kbsystem(4, 0);
qwsystem(18, 0);
return;
}

void viewSub(char *fname, int subno)
{
int i, j, k, barattr, num, rpos, flag, wid1;
int kb, options;
char *buf, *nbuf;
char **lines;
char *tscrn;

if(nosubs)
return;
tscrn = getmem(4000);
memmove(&cp, (subslist + subno - 1), sizeof(PROG));
buf = getmem(cp.len + 1);
buf[cp.len] = EOS;
fp = fopen(fname, "rb");
fseek(fp, cp.loc, SEEK_SET);
fread(buf, cp.len, 1, fp);
fclose(fp);

i = 0;
nbuf = buf;
while(nbuf)
{
if(nbuf = strchr(nbuf, CR))
{
i++;
nbuf += 1;
}
else
break;
}
num = i;
lines = malloc(num * 4);
if(*buf != EOS)
lines[0] = buf;
nbuf = buf;
```

```
for(i = 1; i <= num; i++)
{
nbuf = strchr(nbuf, CR);
if(nbuf != NULL)
*nbuf = EOS;
nbuf += 2;
lines[i] = nbuf;
}
printa(1, 25, 0x30, spaceline);
printa(1, 25, 0x30, "Press ESC or click the right button to quit");

wid1 = 0;
wopen(1, 1, 80, 24, 0, 0x17, "", tscrn, &wid1);
wwrap(wid1, 0);
barattr = 0x17;
rpos = 1;
flag = 1;
options = HM_USEWPRINTS;

while(1)
{
helpmenu(wid1, options, barattr, num, lines, &rpos, &kb, &flag);
if ((flag == 2)&&(kb == ESCAPE_KEY))
break;
i = 0;
mouse(&i, &j, &k);
if(i > 1) //rt button or rt+lf buttons
break;
flag = 0;
}
wclose(wid1);
free(buf);
free(lines);
free(tscrn);
printa(1, 25, 0x30, spaceline);
printa(1, 25, 0x30, curDir);
return;
}


void showhelp(void)
{
int x, y, style, num, key, rpos, flag;
int battr, cattr, dattr;
char *scrn;
char ch;
```

```
int i, abort = 0, helpitems;
char **menu_items;
char helpfile[130];

ch = po.sep;
po.sep = '@';
strcpy(helpfile, startDir);
strcat(helpfile, "\\HELP.HLP");
fp = fopen(helpfile, "rt");
if(fp == NULL)
{
strcpy(inbuf, "HELP.HLP File is missing");
error();
return;
}
fgets(inbuf, 80, fp);
fgets(inbuf, 80, fp);
inbuf[3] = EOS;
helpitems = atoi(inbuf);

menu_items = malloc((helpitems+1) * 4);
for(i = 0; i <= helpitems; i++)
menu_items[i] = malloc(50);
strcpy(menu_items[0], "Help Topics");

for(i = 1; i <= helpitems; i++)
{
strcpy(menu_items[i], "  ");
fgets(inbuf, 80, fp);
inbuf[strlen(inbuf)-1] = EOS;
if(inbuf[0] == '#')
strcpy(menu_items[i], "");
else
strcat(menu_items[i], inbuf);
}
fclose(fp);
scrn = getmem(2000);
y = 1;
x = 58;
style = SNG_BOX | TRANS_SHADOW;
battr = 0x17;
cattr = 0x17;
dattr = 0x7b;
rpos = 1;
flag = 1;
prepSubNames(helpfile);
```

```
for(i = 0; i < numsubs; i++)
{
(subslist + i)->loc += 5;
(subslist + i)->len -= 5;
}
while(1)
{
 popmenu(x, y, style, battr, cattr, dattr, helpitems,
      menu_items, scrn, &rpos, &key, &flag);
 if(!flag)
    continue;
 if(flag == 1 && rpos <= helpitems)
       {
    viewSub(helpfile, rpos);
    flag = 0;
       }
        else
 break;
        }
popclose();
po.sep = ch;
free(scrn);
for(i = 0; i <= helpitems; i++)
free(menu_items[i]);
free(menu_items);
return;
}

int prepSubNames(char *fname)
{
char *buf, *nbuf, *tbuf, temp[subNameLg];
int i, j, k;
unsigned int bsize, cb, blocks;

bsize = 10  * 1024;

if(fname == NULL || noff == 1)
goto quit;
fp = fopen(fname, "rb");
if(fp == NULL)
goto quit;
size = (unsigned int) filelength(fileno(fp));
if(size <= 0)
{
fclose(fp);
goto quit;
```

```
}

blocks = size / bsize;
if(blocks)
buf = getmem(bsize+1);
else
buf = getmem(size+1);
numsubs = 0;
nosubs = 1;
i = 0;
k = 25;
subslist = realloc(subslist, k * sizeof(PROG));

for(cb = 0; cb <= blocks; cb++)
{
fseek(fp, cb * bsize, SEEK_SET);
if(cb < blocks)
{
fread(buf, bsize, 1, fp);
buf[bsize] = EOS;
}
else
{
fread(buf, size - bsize * blocks, 1, fp);
buf[size - bsize * blocks] = EOS;
}
nbuf = buf;
while(1)
{
nbuf = strchr(nbuf, po.sep);
if(nbuf == NULL)
break;
i++;
if(i > k)
{
k += 25;
subslist = realloc(subslist, k * sizeof(PROG));
}
(subslist + i - 1)->loc = nbuf - buf + cb * bsize;
if(i > 1)
(subslist + i - 2)->len = (subslist + i - 1)->loc -
(subslist + i - 2)->loc;

nbuf++;
memmove(temp, nbuf, subNameLg);
for(j = 0; j < subNameLg; j++)
```

```
if(temp[j] == CR || temp[j] == LF || temp[j] == TAB)
  {
temp[j] = EOS;
break;
}
temp[subNameLg-1] = EOS;
strcpy((subslist + i - 1)->name, temp);
}
}
fclose(fp);
(subslist + i - 1)->len = size -  (subslist + i - 1)->loc;
free(buf);
numsubs = i;
if(numsubs <= 0)
goto quit;
nosubs = 0; //there are subs
return numsubs;
quit:
{
subslist = realloc(subslist, 1 * sizeof(PROG));
strcpy(subslist->name, "None");
numsubs = 1;
nosubs = 1;
return 0;
}
}
```

```
// xygraphs.c: This file contains the functions
// related to graphs that will be plotted.

#include "xygraphs.h"

TITLES titles;

void main(int argc, char* argv[])
{
 int i;
 char dataFile[41];

 if(argc == 1)
  {
   printf("\a\nCorrect syntax is: XYGRAPH.EXE <filename>");
   printf("\nFile should be an ASCII file containing");
   printf("\nLine 1: GraphTitle");
   printf("\nLine 2: X-AxisTitle");
   printf("\nLine 3: Y-AxisTitle");
   printf("\nLine 4: XValue1 YValue1");
   printf("\nLine 5: XValue2 YValue2");
   printf("\n      : ...");
   printf("\nLine n+3: XValueN YValueN");
   printf("\n\n\nEnter file name: ");
   gets(dataFile);
}
     else
 {
  strcpy(dataFile, argv[1]);
 }

 if ((i = readDataFromFile(dataFile)) == FALSE)
  {
   printf("Could not read data from data file: %s", dataFile);
   return;
  }
 xptr = &xValues[0];
 yptr = &yValues[0];


 showgraf(titles, xptr, yptr, numPoints);
 _setvideomode(_TEXTC80);
 return;
 }
```

```
void box(float x1, float y1, float x2, float y2)
{
 _moveto_w(x1,y1);
 _lineto_w(x2,y1);
 _lineto_w(x2,y2);
 _lineto_w(x1,y2);
 _lineto_w(x1,y1);
 _setcolor(5);
 _moveto_w(0,y1);
 _lineto_w(0,y2);
 _moveto_w(x1,0);
 _lineto_w(x2,0);
 _setcolor(15);
 return;
}

float max(float *ptr, int numPoints)
{
 float maxx;
 int i;

 maxx = *ptr;
 for (i= 1; i < numPoints+1; ++i)
  {
   if (*ptr > maxx) maxx = *ptr;
   ++ptr;
  }
 return maxx;
}

float min(float *ptr, int numPoints)
{
 float minn;
 int i;

 minn = *ptr;
 for (i = 1; i < numPoints+1; ++i)
  {
   if (*ptr < minn) minn = *ptr;
   ++ptr;
  }
 return minn;
}

void plus(float x, float y, float lgx, float lgy)
{
```

```
 _moveto_w(x-lgx, y);
 _lineto_w(x+lgx, y);
 _moveto_w(x, y-lgy);
 _lineto_w(x, y+lgy);
 return;
}

void showgraf(TITLES titles, float *ptrx, float *ptry,
int numPoints)
{
 float xdiv, ydiv, xmax, ymax;
 float maxx, maxy, minx, miny, lx, ly, hx, hy;
 float dx, dy, py, px, xrange, yrange;
 float vxrange, vyrange, xorg, yorg;
 int i,j,numxdiv,numydiv;
 int lpx, lpy, hpx, hpy;
 float *ptr, *tmpptrx, *tmpptry;
 int xp = 640, yp = 480;

 maxx = max(ptrx,numPoints);
 maxy = max(ptry,numPoints);
 minx = min(ptrx,numPoints);
 miny = min(ptry,numPoints);
 dx = maxx - minx;
 dy = maxy - miny;

 xrange = 4*dx;
 yrange = 4*dy;
 lx = minx + dx/2 - xrange/2;
 hx = minx + dx/2 + xrange/2;
 ly = miny + dy/2 - yrange/2;
 hy = miny + dy/2 + yrange/2;
 xorg = lx + 0.1*xrange;
 yorg = ly + 0.1*yrange;
 vxrange = 0.8*xrange;
 vyrange = 0.8*yrange;
 numxdiv = 10;
 numydiv = 10;
 xdiv = vxrange/numxdiv;
 ydiv = vyrange/numydiv;

 xorg = round(xorg);
 yorg = round(yorg);
 xdiv = round(xdiv);
 ydiv = round(ydiv);
```

```
xmax = xorg+xdiv*numxdiv;
ymax = yorg+ydiv*numydiv;
var[1] = xorg;
var[2] = xmax;
var[3] = yorg;
var[4] = ymax;
_setvideomode(_VRES16COLOR);
while (TRUE)
{
 _clearscreen(_GCLEARSCREEN);
 xorg = var[1];
 xmax = var[2];
 yorg = var[3];
 ymax = var[4];

 if (xorg == xmax)
  {
   printf("\nX min and X max are equal\a");
   printf("\nNo Graph is plotted\a");
   xmax = xorg + 100;
  }

 if (yorg == ymax)
  {
   printf("\nY min and Y max are equal\a");
   printf("\nNo Graph is plotted\a");
   ymax = yorg + 100;
  }

 vxrange = xmax-xorg;
 vyrange = ymax-yorg;
 xdiv = vxrange/numxdiv;
 ydiv = vyrange/numydiv;
 xrange = 1.25*vxrange;
 yrange = 1.25*vyrange;
 lx = xorg - 0.1*xrange;
 hx = lx + xrange;
 ly = yorg - 0.1*yrange;
 hy = ly + yrange;
 _setwindow(1, lx, ly, hx, hy);

 box(xorg, yorg, xmax, ymax);

 lpx = (int) (1 / xrange * xp * (xorg-lx) );
 lpy = (int) (yp - (yorg-ly+vyrange)*yp/yrange);
 hpx = (int) ((xorg-lx+vxrange)*xp/xrange);
```

```
hpy = (int) (yp - (yorg-ly)*yp/yrange);

_setcliprgn(lpx, lpy, hpx, hpy);
tmpptrx = ptrx;
tmpptry = ptry;
_settextposition(5,50);
printf("  X       Y");
for (i = 1; i < numPoints; ++i)
 {
  plus(*tmpptrx, *tmpptry, vxrange / 100, vyrange / 100);
  _settextposition(5+i, 50);
  printf("%-6.2f   %-6.2f",*tmpptrx, *tmpptry);
  if(i > 1) {
  drawline(*(tmpptrx-1), *(tmpptry-1), *tmpptrx, *tmpptry);
  }
  ++tmpptrx;
  ++tmpptry;
 }


_setcliprgn(0, 0, xp, yp);

for (i=1; i < numxdiv; ++i)
 {
  _moveto_w(xorg+vxrange*i/10,yorg);
  _lineto_w(xorg+vxrange*i/10,yorg+vyrange/20);
 }
for (i = 1; i < numydiv; ++i)
 {
  _moveto_w(xorg, vyrange*i/10+yorg);
  _lineto_w(xorg+vxrange/20,yorg+vyrange*i/10);
 }
_settextposition(1,15);
printf("X min          X max          Y min          Ymax");
_settextposition(3,35);
printf(titles.gtitle);
_settextposition(28,5);
printf("%d",(int) xorg);
_settextposition(27,1);
printf("%d",(int)yorg);
_settextposition(28,71);
printf("%d",(int)(xorg+vxrange));
_settextposition(3,1);
printf("%d",(int)(yorg+vyrange));
_settextposition(28,35);
_outtext(titles.xtitle);
```

```
  _settextposition(29,35);
  printf("1 Div = %d", (int)vxrange/10);
  _settextposition(14,1);
  _outtext(titles.ytitle);
  _settextposition(15,1);
  printf("1 Div = \n%d", (int)vyrange/10);
  _settextposition(30,25);
  printf("U - Update view    Q - Quit");

 display();
 while(TRUE)
  {
   sc.c = getkey();
   if ((sc.ch[0] == 0) && (sc.ch[1] == 72)) up();
   if ((sc.ch[0] == 0) && (sc.ch[1] == 75)) left();
   if ((sc.ch[0] == 0) && (sc.ch[1] == 77)) right();
   if ((sc.ch[0] == 0) && (sc.ch[1] == 80)) down();
   if (sc.ch[0] == 13) enter();
   if ((sc.ch[0] >= '0' && sc.ch[0] <= '9') || sc.ch[0] == '.'||
sc.ch[0] == '-' || sc.ch[0] == '+')
    enter();
   if (sc.ch[0] == 'u' || sc.ch[0] == 'U')
break;
   if (sc.ch[0] == 'q' || sc.ch[0] == 'Q' || sc.ch[0] == 27 )
break;
  }
 if (sc.ch[0] == 'q' || sc.ch[0] == 'Q' || sc.ch[0] == 27 )
break;
 }
 return;
}


void display()
{
 int i;
 for (i = 1; i <= NUMVARS; ++i)
  {
   _settextposition(row(i), clm(i));
   printf("%f",var[i]);
  }
 _settextposition(row(pos), clm(pos));
 _settextcolor(fcolor);
 _setbkcolor(bcolor);
 sprintf(numstr,"%f",var[pos]);
 _outtext(numstr);
```

```
 return;
}

int clm(int c)
{
 if (c%COLUMNS == 0)
  return COLUMNS*CLEN;
 else
  return (c%COLUMNS)*CLEN;
}

int row (int c)
{
 return 2;
}

void down()
{
 var[pos] = var[pos] - incr[pos];
 _settextposition(row(pos), clm(pos));
 sprintf(numstr, "%f", var[pos]);
 _outtext(numstr);
 return;
}


void enter()
{
 char newstr1[16];
 char newstr2[15];
 char *ptr1, *ptr2;

 _settextposition(row(pos), clm(pos));
 _settextcolor(fcolor);
 _setbkcolor(bcolor);
 if ((sc.ch[0] >= '0' && sc.ch[0] <= '9') || sc.ch[0] == '.' ||
sc.ch[0] == '-' || sc.ch[0] == '+')
  {
   printf("%c",sc.ch[0]);
   gets(newstr2);
   newstr2[14] = '\0';
   newstr1[0] = sc.ch[0];
   newstr1[1] = '\0';
   strcat(newstr1, newstr2);
   sscanf(newstr1, "%f",&var[pos]);
  }
```

```
  display();
 return;
}

getkey()
{
 union REGS r;
 r.h.ah = 0;
 return int86(0x16, &r, &r);
}


void leave()
{
 _settextposition(row(pos), clm(pos));
 _settextcolor(bcolor);
 _setbkcolor(fcolor);
 sprintf(numstr, "%f", var[pos]);
 _outtext(numstr);
 return;
}

void left()
{
 leave();
 pos = pos - 1;
 if (pos == 0) pos = NUMVARS;
 _settextposition(row(pos), clm(pos));
 _settextcolor(fcolor);
 _setbkcolor(bcolor);
 sprintf(numstr, "%f", var[pos]);
 _outtext(numstr);
 return;
}


void right()
{
 leave();
 pos = pos + 1;
 if (pos > NUMVARS) pos = 1;
 _settextposition(row(pos), clm(pos));
 _settextcolor(fcolor);
 _setbkcolor(bcolor);
 sprintf(numstr, "%f", var[pos]);
 _outtext(numstr);
```

```
 return;
}

void up()
{
 var[pos] = var[pos] + incr[pos];
 _settextposition(row(pos), clm(pos));
 sprintf(numstr, "%f", var[pos]);
 _outtext(numstr);
 return;
}

float round(float x)
{
 float y, f, z;
 int i;
 if (x == 0) return x;
 y = log10(fabs(x));
 i = floor(y);
 if (y < 0)
  f = y + fabs(i);
 else
  f = y - fabs(i);
 f = pow(10,f);
 if (fmod(f,1) < 0.5)
   f = floor(f);
 else
   f = ceil(f);
 z = f*pow(10,i);
 if (x < 0 ) return -z;
 if (x > 0 ) return z;
}

void dispstr(int x, int y, char *ptr)
{
 _settextposition(x,y);
 _outtext(ptr);
}

void dispstrc(int x, int y, int fc, int bc, char *ptr)
{
 int oldfc, oldbc;
 _settextposition(x,y);
 oldfc = _settextcolor(fc);
 oldbc = _setbkcolor(bc);
 _outtext(ptr);
```

```
 _settextcolor(oldfc);
 _setbkcolor(oldbc);
}

void prints(char *ptrs)
{
 while (*ptrs) bdos(0x5,*ptrs++,0);
}


int readDataFromFile(char* dataFile)
// read data from dataFile
// set xptr and yptr pointer to point x and y series
{
FILE* fp;
char inbuf[41];
int i;
char xstr[10];
char ystr[10];

fp = fopen(dataFile, "rt");
if (fp == NULL)
 {
   return FALSE;
 }

// read titles
fgets(titles.gtitle, 40, fp);
fgets(titles.xtitle, 40, fp);
fgets(titles.ytitle, 40, fp);

// read data
i = 0;
while (TRUE)
 {
   fgets(inbuf, 40, fp);
   if(strlen(inbuf) <= 1)
      break;
   else
     {
      sscanf(inbuf, "%s %s\n", xstr, ystr);
      xValues[i] = atof(xstr);
      yValues[i] = atof(ystr);
      i++;
     }
 }
```

```
numPoints = i + 1;
fclose(fp);
return TRUE;
}

void drawline(float x1, float y1, float x2, float y2)
{
_moveto_w(x1,y1);
_lineto_w(x2,y2);
return;
}
```

APPENDIX D

HEADER FILES

```
// const.h: Header file containing Constants
#define M_MSGS 10
#define TOT_MSGS 100
#define MAX_NODES 63
#define MAX_NAME 10
#define MAX_OBJS 60
#define MAX_OVERHEAD 67
#define MAX_MSG_LEN 132
#define YES 1
#define NO 0
#define BUSY 1
#define IDLE 0
#define OVER 0
#define FAILURE 0
#define SUCCESS 1
#define OVERLOAD_ERROR 2
#define FORM_ERROR 3
#define CRC_ERROR 4
#define ACK_ERROR 5
#define ACTIVE 1
#define PASSIVE 0
#define BUS_OFF -1
#define ESCAPE_KEY 27
#define NMENUS 4     // Number of menus
#define  ESC   27    // Escape Key
#define  CR    13    // Carriage Return
#define  LF    10    // Line Feed
#define  TAB    9    // Tab Key
#define  SPACE 32    // Space Bar
#define  EOS  '\0'   //end of string
#define  subNameLg 10

#define  TICKPERSEC 18.2
#define BIOS_DATA ((int far *)(0x400000L))

#define cson _settextcursor(1)
#define csoff _settextcursor(0)
```

```
// protos.h: Header file containing prototypes

void runsim();
void sys_init();
void node_addressing();
int  get_parm();
void msg_cycl();
void prior_assign();
void packs_init(int S);
void packr_init(int R);
void statistics();
int  rand_error();
int  receive(int r_bit, int br, int indr, int nr);
int  get_bit(unsigned char g_val, int bits);
int  transmit(int n);
int  msg_filter(int bit_val, int bm, int indm, int nm);
int  crc_check(int nr);
int  arbitrate();
int  send_overload_frame();
int  send_error_frame(int ne);
void msg_transfer(int mode);
void crc_gen(int Node, int Msg);
void packet_gen(int node_no, int msg_no);
void windowstat();
void windowResults();

// GUI proto types

int  getFileName(void);
int  getFileNameMode1(void);
int  getFileNameMode2(void);
int  prepSubNames(char *fname);
int  prepFileNames(char *mask, int fattr);
int  prepDirNames(char *mask, int fattr);
int  sortCompare(char *str1, char *str2);
void viewSub(char *fname, int subno);
int  setPrinter(void);
int  getString(char *prompt, char *string, int lg, int spaceout);

void search(void);
void printFile(void);
void printResultFile(void);
void fileutil(void);
void editfile(void);
void newfile(void);
void setDirDrive(void);
```

```
void setFilemask(void);
void shell(void);
void exitmm(void);
void error(void);
char *getmem(int size);
void trim(char *string);
int  saveSettings(void);
void readSettings(void);
void setup(void);
void welcome(void);
void waitevent(void);
void showfileinfo(char *fname);
void printStr(char *buf, int nextline);
void popmasks(void);
void defineMasks(void);
void backup(void);
int  filebackup(char *, char *, char *);
void showhelp(void);
```

```
// defs.h: Header file containing data structures
// and global variables used in simulation.

#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <malloc.h>

long busy_time, slack_time, response_time, remote;
long losers, idle_time, errors, msg_time, error_overhead;
int  sample_count, sim_cnt, count, data_frame, standard;
long tic, finish, latency, sample_time, sample_period;
long rand_rate, error_period, random_error;
int  collisions, transmitted, missed, retransmitted;
char bus, bus_flag;
long max_period, min_period, periodic_error;
int  nr, n, m, total_msgs, total_nodes;
int  no_of_receivers, ones, zeros, prv_bit, pos;
int  ack_count, overload_count, crc_count, form_count;
FILE *ip, *op, *wst;
FILE *g1, *g2, *g3, *g4, *g5, *g6, *g7, *g8, *g9;

typedef struct
{
char inputFile[14]; // default input file
long simDuration;   // simulation duration in ms
int  bandwidth;     // to increase simDuration
int  seed;          // seed for random number generator
long t1, t2;        // times for sampling window in ms
int  networkSpeed;  // between 1 and 32000 in kbps
int  samples;       // number of samples to collect stats for
int  errorGenMode;  // 0 = None, 1 = periodic, 2 = random,
    // 3 = periodic+random
int  pErrorRate;    // errors per millisec
int  rErrorRate;    // random error rate between 1 and 32000
    // the larger the rate, the lower number of errors
long simSpeed;      // visual simulation speed
int  sporadicMsgGenMode; // 0 none 1 random
} SP;       //simulation parameters data structure

SP sp;

typedef struct    // formatted packet
 {
```

```
  unsigned char sof;
  unsigned char arb[4];
  unsigned char ctr;
  unsigned char dat[8];
  unsigned char crc[2];
  unsigned char ack;
  unsigned char eof;
  unsigned char isp;
  } PACKET ;

typedef struct
{
 long data_len, period, release, deadline, prior;
 long msg_format, msg_mode, arb_lost, error_flag;
 float  trans_time;
 char msg_name[MAX_NAME];
} MESSAGE;

typedef struct {        // node parameters
  PACKET packs, packr;
  MESSAGE msg[M_MSGS];
  int curr_msg, no_of_msgs;
  char node_name[MAX_NAME];
  int no_of_objs, object[MAX_OBJS];
  unsigned char ovrhd_delim, err_flag, err_delim;
  unsigned char ovrhd_flag,  address;
  char prv_bus, prv_bus_flag;
  int status, bit_val, transfer, receive;
  int dead_count, recv_err_cnt, trans_err_cnt;
  int error_count, lost_count, trans_count;
} NODE;

NODE node[MAX_NODES];

typedef struct
{
   char editor[51];
   char browser[51];
   char dataFilesDir[51];
   char filemask[13];
   char bakupDir[51];
   int  prnport;
   char sep;
   char masks[10][13];
} PO;
```

```
typedef struct
{
char name[subNameLg];
int  loc;
int  len;
} PROG;

PO    po; // prog options
PROG  cp, *subslist;
FILE *fp;

unsigned long size;

int noff, nosubs,numfiles,numdirs,numdrives,
    numsubs,firsttime;

char *fileslist,*dirslist,*savearray,*mbscrn,
     *driveslist,ch,spaceline[80],comm[151],
     inbuf[257],cfilename[51],curDir[129],
     startDir[129],srchstr[21],srchmask[13];
```

```
// gui.h: Header file containing GUI functions

char *menubar[]={
  "\x01\x02""File",
  "\x01\x0c""Utilities",
  "\x01\x18""Simulation",
  "\x01\xfe""Help F1"
  };

char *filemenu[]=
{
"\x01""File Utilities F3",
"\x01""Print File     F2",
"\x01""Dos Shell",
"",
"\x02""eXit           F10"
};

char *utilmenu[]=
{
"\x0a""Set file Mask  F5",
"\x05""Set Dir/Drive",
"\x01""Backup files   F4",
"",
"\x01""String Search  F6",
"\x08""System Info"
};

char *sendrecvmenu[]=
{
"\x01""Run Simulation     F7",
"\x01""Select Input File  F8",
"\x01""Change Parameters  F9",
"\x04""Real-Time Checks",
"\x06""View Message Set Details",
"",
"\x0f""View Results: Text",
"\x0f""View Results: Graphs",
"\x01""Print Results"
};

char **menutext[NMENUS] = {filemenu, utilmenu, sendrecvmenu};

int  msize[NMENUS] = {5, 6, 9, 0};

char *menuscrn;
```

```
int  menuid;
char *fake;
int  barattr, barrev, charen, chardis;
int  ahot, aen, adis, abdr;
int  style, options, menu, flag, item, kb;

extern void comInit(void);
extern void sendFile(char *fname, unsigned int mode);
extern void recvFile(void);
extern void search(void);
extern void printFile(void);
extern void printResultFile(void);
extern void fileutil(void);
extern void editfile(void);
extern void newfile(void);
extern void setDirDrive(void);
extern void setFilemask(void);
extern void setup(void);
extern void sendSub(void);
extern void subprograms(void);
extern void shell(void);
extern void exitmm(void);
extern void subprograms(void);
extern char *getmem(int);
extern void showhelp(void);
```

```
// xygraph.h: Header file for xygraph.c
// This file contains graphic functions.

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <graph.h>
#include <dos.h>
#include <math.h>

#define MAXPOINTS 50
#define TRUE 1
#define FALSE 0

#define NUMVARS 4
#define ROWS 1
#define COLUMNS 4
#define CLEN 15

typedef struct
 {
   char gtitle[41];
   char xtitle[41];
   char ytitle[41];
 } TITLES;

int pos = 1;
char numstr[15];
char inbuf[100], xstr[15], ystr[15];
float var[NUMVARS+1];
float incr[NUMVARS+1] = {0, 50, 50, 50, 50};
int fcolor = 2, bcolor = 7;
union scan
 {
  int c;
  char ch[2];
 } sc;

float xValues[MAXPOINTS];
float yValues[MAXPOINTS];
float *xptr, *yptr;
int numPoints;

void showgraf(TITLES titles, float *ptrx, float *ptry, int numPoints);
void box(float x1, float y1, float x2, float y2);
float max(float *ptr, int numPoints);
```

```
float min(float *ptr, int numPoints);
void plus(float x, float y, float lg, float lgy);
getkey();
void display();
void prints(char *ptrs);
int  clm(int c);
int  row(int c);
void left();
void right();
void up();
void down();
void enter();
void leave();
float round(float x);
void dispstrc(int x, int y, int fc, int bc, char *ptr);
void dispstr(int x, int y, char *ptr);
int  readDataFromFile(char*);
void drawline(float x1, float y1, float x2, float y2);
```

APPENDIX E

INPUT FILES

INPUT FILE I

```
engine 3 3
MSG11    10    1    31    8   1 1
MSG12    50    0    30    8   0 1
MSG13    250    0   29    8   1 1
2
0
0
torque 1 2
MSG21    10    0    28    8   1 1
0
0
```

INPUT FILE II

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
```

INPUT FILE III

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
```

```
eng_fluid 1 1
MSG121    1000    0 13    8   1 1
0
eng_temp 1 1
MSG131    1000    0 12     8   1 1
0
```

INPUT FILE IV

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
```

```
eng_fluid 1 1
MSG121   1000   0 13   8  1 1
0
eng_temp 1 1
MSG131   1000   0 12    8  1 1
0
eng_hrs 1 1
MSG141   10   0   11   8  1 1
0
pto_def 1 1
MSG151   100   0  10    8  1 1
0
idle_pto 1 1
MSG161   1000   0 9    8  1 1
0
speed 1 1
MSG171   100   0  8   8  1 1
0
calib 1 1
MSG181   10   0   7   8  1 1
0
miles 1 1
MSG191   10   0   6   8  1 1
0
fuel 2 1
MSG201   200   0  5   8  1 1
MSG202   10   0   4   8  1 1
0
```

INPUT FILE V

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
```

```
eng_fluid 1 1
MSG121   1000   0 13   8  1 1
0
eng_temp 1 1
MSG131   1000   0 12    8  1 1
0
eng_hrs 1 1
MSG141   10   0   11   8  1 1
0
pto_def 1 1
MSG151   100   0  10    8  1 1
0
idle_pto 1 1
MSG161   1000   0 9    8  1 1
0
speed 1 1
MSG171   100   0  8   8  1 1
0
calib 1 1
MSG181   10   0   7   8  1 1
0
miles 1 1
MSG191   10   0   6   8  1 1
0
fuel 2 1
MSG201   200   0  5   8  1 1
MSG202   10   0   4   8  1 1
0
ind 2 1
MSG211   100   0   3   8  1 1
MSG212   200   0   2   8  1 1
0
tire 1 1
MSG221   10000   0 37   8  1 1
0
amby 1 1
MSG231   1000   0  38   8  1 1
0
exhst 1 1
MSG241   1000   0  39   8  1 1
0
power 1 1
MSG251   1000   0  40   8  1 1
0
fluids 1 1
MSG261   1000   0  41   8  1 1
```

```
0
dash 1 1
MSG271   10000   0 43    1  1 1
0
water 1 1
MSG281   10000   0 45   7  1 1
0
diag 2 1
MSG291   600   0   46  3  1 1
MSG292   700   0   47   3  1 1
0
ind1 1 1
MSG301   800   0   36   8  1 1
0
```

INPUT FILE VI

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
```

```
eng_fluid 1 1
MSG121   1000   0 13   8  1 1
0
eng_temp 1 1
MSG131   1000   0 12    8  1 1
0
eng_hrs 1 1
MSG141   10   0   11   8  1 1
0
pto_def 1 1
MSG151   100   0  10    8  1 1
0
idle_pto 1 1
MSG161   1000   0 9    8  1 1
0
speed 1 1
MSG171   100   0  8   8  1 1
0
calib 1 1
MSG181   10   0   7   8  1 1
0
miles 1 1
MSG191   10   0   6   8  1 1
0
fuel 2 1
MSG201   200   0  5   8  1 1
MSG202   10   0   4   8  1 1
0
ind 2 1
MSG211   100   0   3   8  1 1
MSG212   200   0   2   8  1 1
0
tire 1 1
MSG221   10000   0 37   8  1 1
0
amby 1 1
MSG231   1000   0  38   8  1 1
0
exhst 1 1
MSG241   1000   0  39   8  1 1
0
power 1 1
MSG251   1000   0  40   8  1 1
0
fluids 1 1
MSG261   1000   0  41   8  1 1
```

```
0
dash 1 1
MSG271   10000   0 43    1  1 1
0
water 1 1
MSG281   10000   0 45   7  1 1
0
diag 2 1
MSG291   600   0   46  3  1 1
MSG292   700   0   47   3  1 1
0
ind1 1 1
MSG301   800   0   36   8  1 1
0
ind2 1 1
MSG311   800   0   35   8  1 1
0
ind3 1 1
MSG321   700   0   34   8  1 1
0
ind4 1 1
MSG331   600   0   33   8  1 1
0
ind5 1 1
MSG341   500   0   32   8  1 1
0
ind6 1 1
MSG351   400   0   63   8  1 1
0
```

INPUT FILE VII

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
eng_fluid 1 1
```

```
MSG121   1000   0 13   8  1 1
0
eng_temp 1 1
MSG131   1000   0 12    8  1 1
0
eng_hrs 1 1
MSG141   10   0   11   8  1 1
0
pto_def 1 1
MSG151   100   0  10    8  1 1
0
idle_pto 1 1
MSG161   1000   0 9    8  1 1
0
speed 1 1
MSG171   100   0  8   8  1 1
0
calib 1 1
MSG181   10   0   7   8  1 1
0
miles 1 1
MSG191   10   0   6   8  1 1
0
fuel 2 1
MSG201   200   0  5   8  1 1
MSG202   10   0   4   8  1 1
0
ind 2 1
MSG211   100   0   3   8  1 1
MSG212   200   0   2   8  1 1
0
tire 1 1
MSG221   10000   0 37   8  1 1
0
amby 1 1
MSG231   1000   0  38   8  1 1
0
exhst 1 1
MSG241   1000   0  39   8  1 1
0
power 1 1
MSG251   1000   0  40   8  1 1
0
fluids 1 1
MSG261   1000   0  41   8  1 1
0
```

```
dash 1 1
MSG271   10000   0 43    1  1 1
0
water 1 1
MSG281   10000   0 45   7  1 1
0
diag 2 1
MSG291   600   0   46  3  1 1
MSG292   700   0   47   3  1 1
0
ind1 1 1
MSG301   800   0   36   8  1 1
0
ind2 1 1
MSG311   800   0   35   8  1 1
0
ind3 1 1
MSG321   700   0   34   8  1 1
0
ind4 1 1
MSG331   600   0   33   8  1 1
0
ind5 1 1
MSG341   500   0   32   8  1 1
0
ind6 1 1
MSG351   400   0   63   8  1 1
0
ind7 1 1
MSG361   300   0   1   8  1 1
0
ind8 1 1
MSG371   30   0   1   8  1 1
0
ind9 1 1
MSG381   40   0   1   8  1 1
0
ind10 1 1
MSG391   50   0   1   8  1 1
0
```

INPUT FILE VIII

```
engine 3 3
MSG11   10   1   31   8  1 1
MSG12   50   0   30   8  0 1
MSG13   250   0  29   8  1 1
2
0
0
torque 1 2
MSG21   10   0   28   8  1 1
0
0
trans1 1 1
MSG31   10   0   27   8  1 0
0
trans2 3 1
MSG41   10   0   26   8  1 1
MSG42   100   0  25   8  1 1
MSG43   1000   0 24   8  1 1
0
brake 2 1
MSG51   100   0  23   8  1 1
MSG52   1000   0 22   8  1 1
0
retarder 2 1
MSG61   100   0  21   8  1 1
MSG62   1000   0 20   8  1 1
0
brk_ctrl 1 1
MSG71   50   0   19  8  1 1
0
axle 2 1
MSG81   30   0   18   8  1 1
MSG82   1000   0 17   8  1 1
0
eng_con 1 1
MSG91   5000   0 16   8  1 1
0
trans_con 1 1
MSG101   10   0  15   8  1 1
0
retr_con 1 1
MSG111   10   0  14   8  1 1
0
eng_fluid 1 1
```

```
MSG121   1000   0 13   8  1 1
0
eng_temp 1 1
MSG131   1000   0 12    8  1 1
0
eng_hrs 1 1
MSG141   10   0   11   8  1 1
0
pto_def 1 1
MSG151   100   0  10    8  1 1
0
idle_pto 1 1
MSG161   1000   0 9    8  1 1
0
speed 1 1
MSG171   100   0  8   8  1 1
0
calib 1 1
MSG181   10   0   7   8  1 1
0
miles 1 1
MSG191   10   0   6   8  1 1
0
fuel 2 1
MSG201   200   0  5   8  1 1
MSG202   10   0   4   8  1 1
0
ind 2 1
MSG211   100   0   3   8  1 1
MSG212   200   0   2   8  1 1
0
tire 1 1
MSG221   10000   0 37   8  1 1
0
amby 1 1
MSG231   1000   0  38   8  1 1
0
exhst 1 1
MSG241   1000   0  39   8  1 1
0
power 1 1
MSG251   1000   0  40   8  1 1
0
fluids 1 1
MSG261   1000   0  41   8  1 1
0
```

```
dash 1 1
MSG271   10000   0 43    1  1 1
0
water 1 1
MSG281   10000   0 45   7  1 1
0
diag 2 1
MSG291   600   0   46  3  1 1
MSG292   700   0   47   3  1 1
0
ind1 1 1
MSG301   800   0   36   8  1 1
0
ind2 1 1
MSG311   800   0   35   8  1 1
0
ind3 1 1
MSG321   700   0   34   8  1 1
0
ind4 1 1
MSG331   600   0   33   8  1 1
0
ind5 1 1
MSG341   500   0   32   8  1 1
0
ind6 1 1
MSG351   400   0   63   8  1 1
0
ind7 1 1
MSG361   300   0   1   8  1 1
0
ind8 1 1
MSG371   30   0   1   8  1 1
0
ind9 1 1
MSG381   40   0   1   8  1 1
0
ind10 1 1
MSG391   50   0   1   8  1 1
0
ind11 1 1
MSG401   60   0   1   8  1 1
0
ind12 1 1
MSG411   70   0   1   8  1 1
0
```

```
ind13 1 1
MSG421   80   0   1   8  1 1
0
ind14 1 1
MSG431   90   0   1   8  1 1
0
ind15 1 1
MSG441   100   0   1   8  1 1
0
```

VITA

Mujibur Rehman Mohammad

Candidate for the Degree of

Master of Science

Thesis: A TIME-DRIVEN DISCRETE-EVENT SIMULATOR FOR CONTROLLER AREA NETWORKS

Major Field: Computer Science

Biographical Data:

Personal Data: Born in Nellore, Andhra Pradesh, India, on May 22, 1969, the son of M. A. Khaiyoom and Ghousinnisa Begum.

Education: Graduated from C.A.M. High School, Nellore, India, in 1984; graduated from V.R. College, Nellore, India, 1986, received Bachelor of Engineering in Electrical Engineering from R. V. College of Engineering, Bangalore, India in February 1991. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1994.

Experience: Computer Operator, Computing and Information Services, Oklahoma State University, August, 1993 to December 1994. Graduate Trainee Engineer Indian Telephone Industries, Bangalore, India, March, 1990 to July, 1990.