SIMULATION OF SAE J1587 STANDARD FOR ELECTRONIC DATA INTERCHANGE BETWEEN MICROCOMPUTER SYSTEMS IN HEAVY-DUTY VEHICLE APPLICATIONS

By

VISWANATHAN ANURADHA

Bachelor of Engineering

Bangalore University

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
July, 1995

SIMULATION OF SAE J1587 STANDARD FOR ELECTRONIC DATA INTERCHANGE BETWEEN MICROCOMPUTER SYSTEMS IN HEAVY-DUTY VEHICLE APPLICATIONS

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

[illegible]

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my graduate advisor Dr. Mitchell L. Neilsen for his guidance, encouragement and help for the completion of my thesis work. Without his constant support, supervision and ideas, this thesis would have been impossible. I would like to thank Dr. K.M. George for serving on my graduate committee and providing me with support for my thesis. I would like to thank Dr. Marvin Stone for providing me with all the technical details concerning the protocol. Every discussion with him has been a pleasant experience. I would also like to thank my committee for this research opportunity and their generous financial support.

I would also like to express my special appreciation to my husband, Ram, for his strong encouragement at times of difficulty, love and understanding throughout the whole process. Thanks also goes to my parents without whom I would never have come here. Thanks also goes to Padhu, Montserrat, Ashwini and grandparents for their support and encouragement.

Finally, I would like to thank Dr. J.D. Carlson from the Department of Biosystems and Agricultural Engineering for supporting me during these two years of study.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

The functionality of automotive systems has been improving with various additions such as real-time Electronic Control Units (ECUs) for engine management, anti-lock braking, and various other applications. Thus, customer expectations include the assurance that products will interface properly with one another and the ability to diagnose and troubleshoot electronic systems. In addition, components should protect themselves from damage, should provide self-diagnosis, perform efficiently and interface to standard service tools. A system must be properly designed and implemented to satisfy these growing demands.

Two of the key factors in each design change are cost and performance. Currently, vehicles are designed to achieve both of these objectives. There has been an increase in the number of electronically controlled systems found in automobiles during the past several years. Until recently, all electronic systems worked independently. The combined system performance can be optimized, without introducing the additional hardware burden for sensors and actuators, by constructing a serial communication network (often called multiplexing) [KIEN88]. The term 'multiplexing' as defined by the SAE Multiplexing and Data Communications subcommittee is *a system transmitting several different messages on the same circuit or channel.* In real-time systems, deadlines are critical and they must be met on time to prevent a catastrophe from happening due to delayed delivery of a message. Data transmission in a network is affected by the distribution of the traffic in the network. The data transmission is also subjected to timing delays due to the latency of messages in the network. Some of the other problems include messages that may be lost due to buffer saturation or corrupted by noise in the network. All of these factors will further degrade the performance of the network.

A major concern is the performance of such a network under heavy load. This means that as the message rate increases, the throughput and delay should remain stable. The scheduling of such messages is complex in a real-time system where deadlines must be met.

An industry standard is required to ensure interoperability. On the basis that one protocol can meet the needs, there are many things to be gained by using a common standard including common diagnostic tools and support, reduced component design engineering time, enhanced supplier support and innovation, enhance manufacturing, reduced complexity, and fluent evolution of future in-vehicle networks.

1.1 SAE J1587 Standard Protocol

The J1587 Protocol defines communication requirements in a real-time network that runs at a baud rate of 9600 bits per second. The network is used primarily for data communications, and has a unique way of handling collisions. All J1587 transmitters monitor the message identification portion of their message to determine if another transmitter has attempted to gain access to the bus at the same time. If a transmitter detects a collision, the transmitter relinquishes control of the bus after completing the transmission of the current character. It resolves the collision by backing off and recalculating a new bus access time based on message priority. Finally, if there have been three consecutive collisions, it produces a pseudo-random number for the priority. Then, the winner may attempt to regain access to the bus after the bus access time has elapsed. The fastest update period is 100 ms. The J1587 Protocol is discussed in Chapter III.

1.2 Problem Statement

Although several studies on LANs and CANs have been made, no study has been reported in the area of the J1587 Protocol. The purpose of this research is to find an efficient way to analyze the periodic and sporadic message sets that the J1587 Protocol application presents. A comprehensive J1587 Simulation Model has been developed to test and analyze the network performance. The distinctive feature of the simulator is that it constructs a bitwise trace of the protocol. Also, functions of error management and fault confinement have been included to analyze message error and node failure overheads. The implementation details of the simulation, and the performance evaluation are discussed in Chapter IV. Finally, the thesis concludes with a summary and a brief discussion of future research in Chapter V.

CHAPTER II

LITERATURE REVIEW

In-vehicle networking or multiplexing primarily addresses two areas - information sharing and electronics control. Properly designed, the benefits which may be obtained are both large and measurable. The benefits may be measured in system cost reduction and performance enhancement, including enhanced diagnostics, reduction of sensor count, distributed control, and total wire reduction. In-vehicle networking may be separated into three areas: real-time control, status-type data sharing, and body electronics control [PHAI86]. Although in-vehicle networking applications really have many of the same requirements, the differences lie primarily in the rate at which the action occurs.

To effectively network the vehicle and allow inter-communication between the application types, one positive approach is to implement a common protocol throughout the vehicle. This allows straightforward communication between the different application buses and more importantly allows one diagnostic tool to service the vehicle, compared to possibly three tools to service three protocols in the same vehicle. Several underlying protocols have been implemented. Some of the existing in-vehicle networks include the C^2 D Network and the Controller Area Network.

2.1 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 [MIES86]. Control Multiplexing affects the base cost of the vehicle. Data Communications interconnects the different modules. It doesn't affect the base wiring of the vehicle. By vehicle multiplexing there will be dramatic savings in the amount of wiring and complexity. Data Multiplexing results in elimination of redundant hardware and firmware. The C^2D network uses a Carrier Sense Multiple Access (CSMA) protocol. The identification byte of each message is unique. The message ID is transmitted first, followed by the message data. The data is followed by a message CRC and is concluded by 10 bits which defines the end of transmission. UART-NRZ was chosen as the bit encoding technique [MIES86]. Typically 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. The Arbitration Detector checks if there is any arbitration taking place between two messages; i.e., if two messages want to transmit at the same time, the Arbitration Detector allows transmission on a first-come-first-serve basis. On the other hand, the Collision Detector, allows a message with the highest priority to transmit. This network has an advantage that the modules can be added or deleted without affecting the network.

2.2 Controller Area Network (CAN)

CAN is a real-time serial communications network. CAN is also used as an in-vehicle network which was developed to resolve the spacing problem raised by wiring, control units, sensors and other components. In 1985, Robert Bosch and Intel agreed to develop CAN. By 1987, the first sample of CAN was produced. CAN includes various properties such as prioritized bitwise arbitration for fast transmission of high priority messages with a latency time of 150 microseconds, high transmission rates in the range of 1 Mbps for a bus length of 40 m, an open system which allows a designer to add or delete any number of nodes without a change in the underlying software or hardware of any node, data consistency, multicast reception, multi-mastership (any node may start transmitting when the bus is free), automatic retransmission of corrupted messages , error detection and error signaling. It has high reliability, low latency, minimum CPU burden for communication, maximum transparency and data consistency.

There are a few CAN simulation packages currently available. Philips produced NetSim, a PC-based simulation package for which a CAN has to be described in terms of the 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 has provided an on-board CAN simulator from which performance measures can be obtained.

At Oklahoma State University, some research has been completed in the area of CAN simulation. A bitwise simulation of the CAN was implemented [NATA93]. The other research completed on CAN at Oklahoma State University analyzed the performance of CAN under asymmetric traffic loads [JEAN94].

CHAPTER III

OVERVIEW OF J1587 STANDARD

3.1 Description of the J1708 and J1587 Standard

The SAE Recommended Practice defines a recommended practice for implementing a bi-directional, serial communication link among modules containing microcomputers. The SAE J1708 document defines those parameters of the serial link that relate primarily to hardware and basic software compatibility such as interface requirements, system protocol, and message format.

The purpose of this standard is to define a general-purpose serial data communication link that can be utilized in heavy-duty vehicle applications. It is intended to serve as a guide toward standard practice to promote serial communication compatibility among microcompter-based modules. The primary use of the general-purpose communications link is expected to be the sharing of data among stand-alone modules to cost effectively enhance their operation. Communication links used to implement functions that require a dedicated communication link between specific modules may deviate from this document.

The J1587 Protocol is a serial communications protocol which efficiently supports distributed real-time control. Its domain of application includes trucks and buses. In vehicle electronics, engine control units, sensors, anti-skid systems, and various other systems can be connected using this protocol.

The J1587 Protocol is a multi-master protocol where messages are transmitted serially. Contention between masters is resolved by using a backoff and retransmission protocol. After two collisions in a row, the protocol uses a pseudo-random number to backoff and retransmit again to hopefully avoid a third collision.

J1587 uses a modified RS485 differential bus (2 lines) with a baud rate of 9600 bps. It can accommodate at least 20 nodes and uses random, destructive and byte-wide network access arbitration. At the data link layer, it uses a standard UART device.

J1587 is a open network where modules can be added or deleted from the network without affecting the operation of the network. J1587 has short transmission times and the fastest update period is 100 ms [SAE93].

Unlike many serial communication protocols, the J1587 message contains no information relating to the destination address. It is sent globally to all other nodes. The message contains an identifier which indicates the type of information contained in the message. This has several important implications. First of all, more nodes can be added or removed from the network without any change in the software. Secondly, each node can decide on the basis of the type of message whether the message is of interest to that particular node. Multicasts to many nodes are therefore inherent in this system and the data will be consistent in that either all or none of the nodes will receive the message [JORD88].

3.2 Message Priority Assignment

All messages are assigned a priority from 1 to 8 as indicated in Table 1:

Priority	Message Assignment
1 and 2	Reserved for messages that require immediate access to the bus.
3 and 4	Reserved for messages that require prompt access to the bus in order to prevent severe mechanical damage.
5 and 6	Reserved for messages that directly affect the economical or efficient operation of the vehicle.
7 and 8	All other messages.

Table 1: Message Priority Assignment

3.3 Message Format

A message appearing on the communication bus consists of the following fields as shown in Figure 1:

a. Message Identification Character (MID)

b. One or More Parameters

c. Checksum

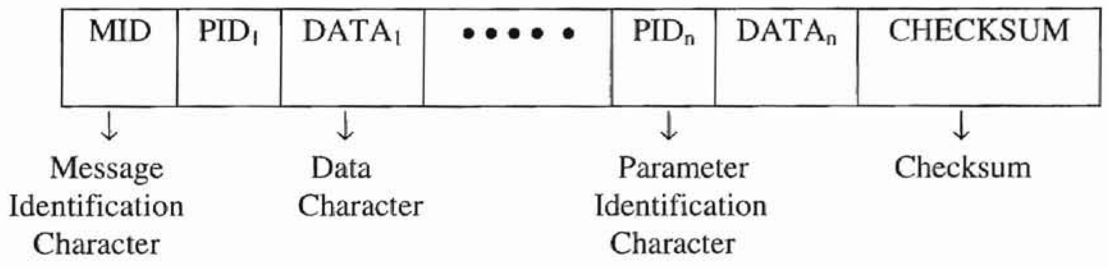

Figure 1: Message format

The message length can vary from 4 - 21 characters. Each character begins with a start bit which is logical bit of 0, and ends with a stop bit of 1. Thus, each character is 10 bits long. A message shall always be preceded by an idle state of duration equal to or greater than the bus access time.

3.3.1 Message Identification Character(MID)

The first character of every message is an MID. The permitted range of MIDs includes the values from 0 to 255. The message identification character is determined by the transmitter category which is predefined.

These assignments have been made to accommodate existing systems, or systems that may presently be under development, and to avoid conflicts which otherwise might arise if indiscriminate use of MIDs were permitted. No two transmitters in the system shall have the same MID.

3.3.2 One or More Parameters

The first character of every parameter is the parameter identification character (PID). The 8 bit parameter identification character is given a value from 0 to 255. Assignment of a PID to a parameter is based on the number of data characters required by the parameter. PIDs from 0 to 127 are allocated to parameters using a single data character to represent its value. This single data character follows the PID. PIDs from 128 to 191 are allocated to double data character parameters. The two data characters follow the PID. Parameters requiring more than two data characters and parameters requiring varying numbers of data characters are allocated the PIDs from 192 to 253. The number of data characters used is contained in the first character after the PID. Data characters are characters that convey the intelligence of the message.

3.3.3 Checksum

The last character of each message is the two's complement form of the sum of the MID and the data characters. A more detailed specifications of the message format for J1587 in Backus-Naur Form (BNF) is given in Appendix B.

3.4 Message Length

Total message length, including MID and checksum, shall not exceed 21 characters.

3.5 Requirements of J1587 Network

A J1587 network should have a modular system concept, configuration flexibility, error detection and error handling schemes. An overall optimization layer should be added to enhance the performance of the network. By dividing the entire system into manageable parts the whole system can be efficiently managed. A J1587 network must be flexible enough to add or delete any number of stations without affecting the underlying hardware or software and application layer.

CHAPTER IV

SIMULATION MODEL

Complex communication protocols ensure error-free exchange of information between communicating entities. From specification to implementation, the different steps in development must be verified for error-free transmission of information.

Modeling techniques of various types can be used to represent the architecture and behavior of communications protocols. Models typically fall into two categories, analytical ones and experimental ones. Experimental models can be implemented using simulation languages and tools. Such models can be used at various stages in the network development lifecycle: requirements analysis during the specification phase, architecture assessment, protocol validation and tuning, and fault diagnosis during network operation. Hence, the need for a simulator. Simulators can be used for productivity improvement studies and management decision support when complex processes are involved. Network simulators can satisfy these requirements.

A computer simulation can be defined as the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. As simulation is a powerful tool to evaluate the performance of any network, J1587 can also be simulated using a computer program. Simulation aids in predicting changes in network performance and comparing alternate designs. Simulation also provides information about various network parameters. A simulation analysis is developed to study the behavior of the protocol with varying system characteristics.

The objective of this research is to analyze the performance of J1587. The performance characteristics include network load, average response time, number of missed deadlines, number of errors, and number of retransmissions. It also features statistical summaries for each node in the network and run-time output. This simulation model can be used to help set standards for the J1587 protocol. To effectively network the vehicle and allow inter-communication between the application types, one positive approach is to implement a common protocol throughout the vehicle. This allows for a straightforward communication between the different applications and the use of a single diagnostic tool to service the vehicle.

Simulation offers a flexible approach for performance evaluation of computer networks. A detailed modeling of J1587 is useful to explore the various design aspects. It also aids in predicting changes in network performance, and comparing alternate designs. Analytical and graphical results can aid the network designer in creating a prototype model. The major drawback is the inability to predict system reliability.

Various modeling approaches, including queuing models, Petri nets, and finite state machines, have been used in the past. A queuing network model does not represent the protocol aspects of the J1587, while the finite state machine model cannot handle the topological features of J1587. Petri net models can be used to verify safety properties of the J1587 protocol. A model of the J1587 protocol is presented in Figure 2. It requires a few assumptions and approximations of the network details.

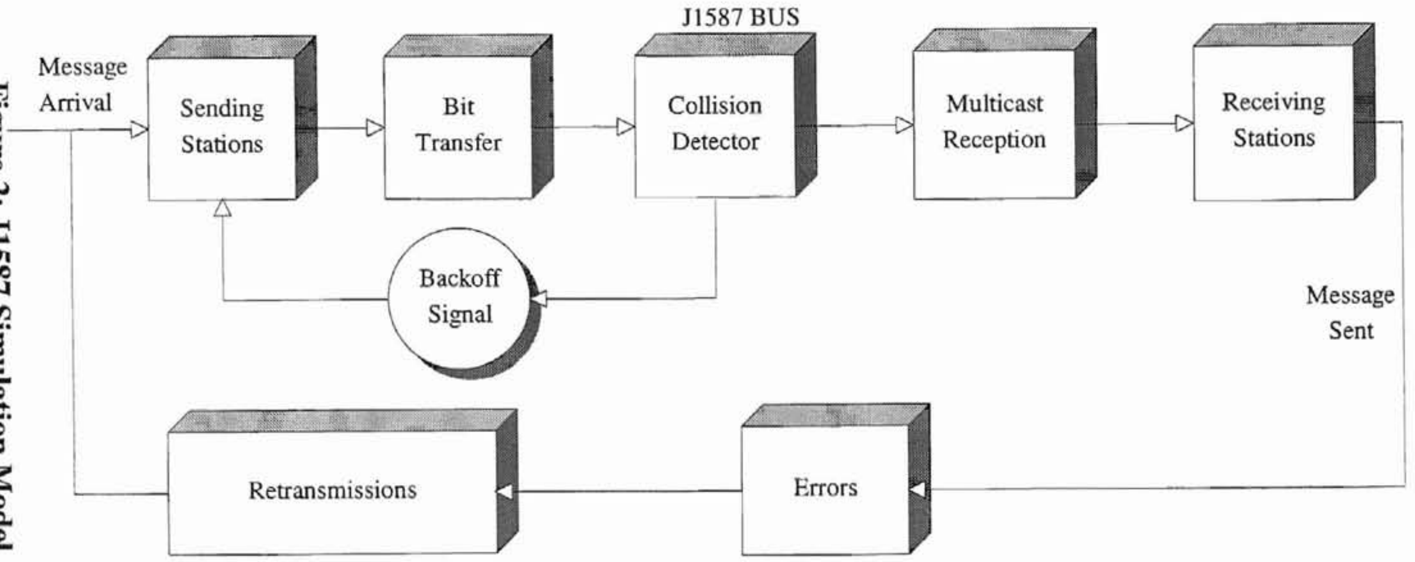

Figure 2: J1587 Simulation Model

Assumptions made include the following:

1. Stations and bus failures are not considered.
2. The simulation results of a small model can be applied to a large model.
3. Message arrival is uniformly distributed.
4. Errors are generated using a uniform random distribution.
5. The priority assigned to a message depends on the first parameter identification character of the message.

4.1 DESIGN

The program design was completed in three phases. The three phases are specification of the protocols, specification of the topology, and specification of the nodes. The first phase was to make a detailed study of the J1587 protocol. The *protocol specification* includes the rules of communication dictated by the protocol within the network. This part forms the core of the simulation, as it represents the flow of control within the simulation program itself. Some of the important algorithms to be studied include the collision detection and backoff algorithm, message transfer, error detection, error signaling and retransmission.

J1587 is a real-time network protocol. A real-time environment is one in which responses to events should occur before a deadline. The J1587 is a hard real-time environment protocol where deadlines must be met, since the failure to signal a braking action in a automobile could be fatal. One of the major hurdles in achieving system reliability, in such hard real-time systems, is finding an efficient way to schedule the events. The following discussion provides the various analogies that can be related from the typical real-time computer systems to J1587. First of all, a non-preemptive process scheduling aptly represents J1587 messages since they cannot be removed from the bus once they are placed on it. Secondly, a uniprocessor machine can be easily viewed as the single channel of communication that the J1587 adopts for message transfer. Scheduling overhead is assumed to be negligible in a real-time computer system. Also, exclusive access to the J1587 bus is guaranteed once the bus contention is resolved. Finally, the processes arriving in a computer system can be readily equated to the messages generated in J1587.

The J1587 messages can be either periodic or sporadic. A periodic message is one that is generated repetitively in fixed time intervals. A typical periodic message M_p is defined as $M_p = (c,p)$ where ‘c’ is the communication cost, and ‘p’ is the period. A message M_p arriving at time t_k has the following rules of generation:

- the (k + 1)-th generation of message M_p will occur at time $t_{k+1} = t_k + p$
- the k-th transfer of the message M_p cannot start before t_k and must be completed no later than its deadline $t_k + p$. That is, the transmission time needs to be in the interval $(t_k , t_k + p)$.

A typical sporadic message is one that is generated in response to an internal or external event. A sporadic message M_s is defined as $M_s = (c,p)$ where ‘c’ is the communication cost, and ‘p’ is the least interval of time before the next generation of such a message. A message M_s arriving at time t_k has the following rules of generation:

- the $(k + 1)$-th generation of message M_s will occur no earlier than $t_k + p$;

  that is $t_{k+1} >= t_k + p$.

- the k-th transfer of the message M_s cannot start before t_k and must be completed no later than its deadline $t_k + p$ [JEFF91].

As the error checking and message transfer operations are bitwise, a bit-by-bit simulation methodology is used (i.e., bit transfers are simulated instead of message transfers).

The second phase involves the *topological module specification*. This essentially determines the physical layout and the physical transmission characteristics under which the network operates. The layout specification is simply the way in which all nodes on the network are connected to the single bus as in Figure 2. The bus topology of the J1587 offers multi-mastership and multicast reception. Thus, any node that has a message to be transmitted simply transmits it, bit-by-bit, on the bus. If a collision occurs, the bus access time is calculated using the back-off algorithm as stated earlier, and then they are retransmitted again. All nodes receive all messages in the simulation because the mask register functionality does not affect the network performance. The most important physical characteristic of the J1587 medium is the baud rate. A J1587 bus has a transfer rate of 9600 bps. The unit of time in the simulation is assumed to be one bit time; that is, the time taken to transit a single bit. In physical terms, one bit time is $104.17\mu s \pm 0.5\%$ for a 9600 bps bus. All times within the simulation are calculated in bit time. The electrical characteristics of J1587 are significant only with respect to voltage fluctuations that result in error conditions.

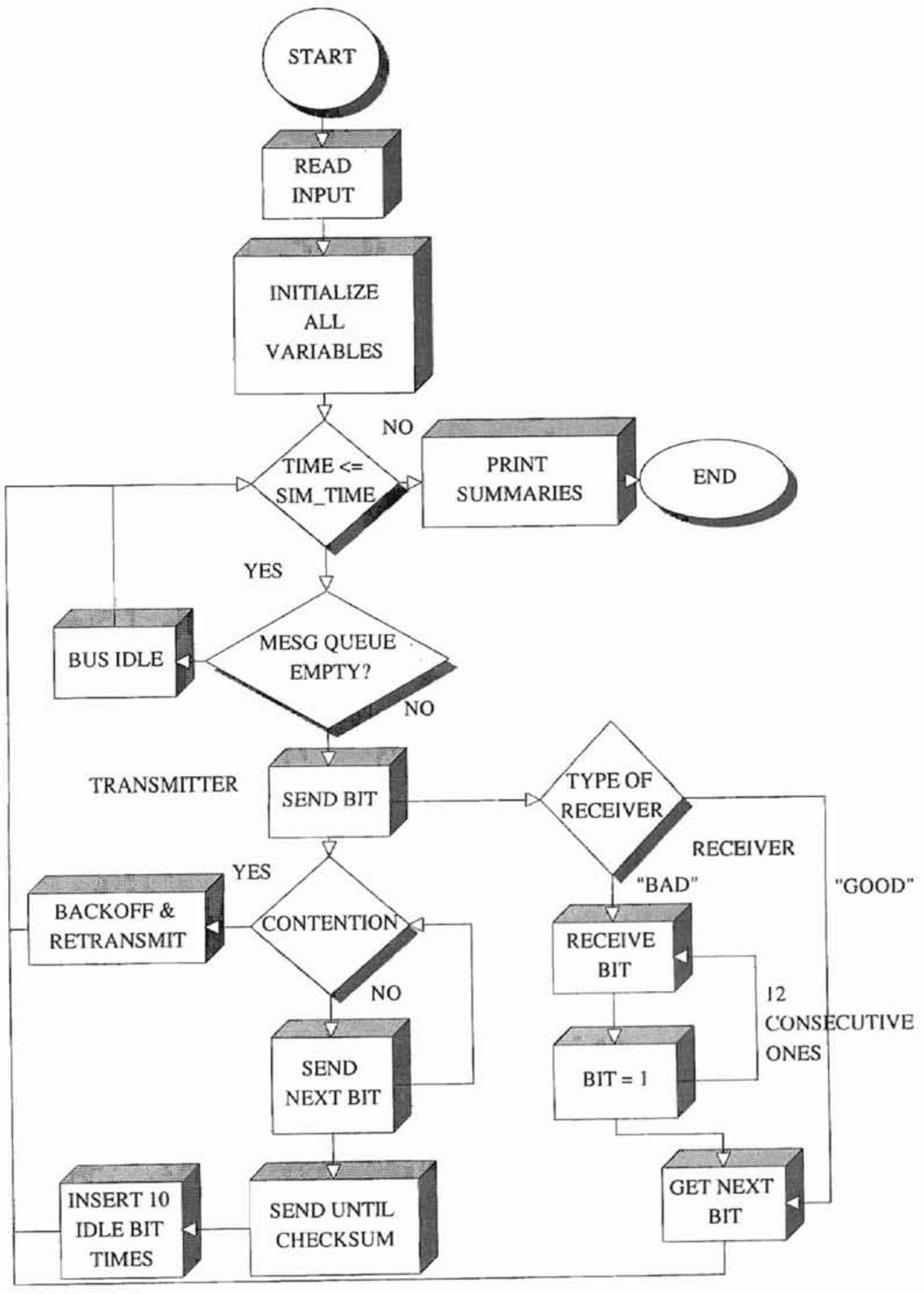

Figure 3: Bit Transfer Flow Diagram

The final phase of the design is the *node module specification*. This includes the specifications of attributes of all nodes connected to the bus. The primary goals of a node are message generation, message transfer, and message reception. All other station details are less important. The following are the main features to look for in a node. The first feature is the type of messages it generates, whether periodic or sporadic. All sporadic messages arrive at their arrival rates uniformly. The second feature in a node is its message characteristics. This includes the message, message release times, and message deadline, and message arrival rates for sporadic messages.

4.2 Implementation

J1587 is a dedicated network being used for specific real-time applications. It has its own distinct standards that define its operation. A simulation package is developed using a bottom-up design. The J1587 simulator is coded in the C Programming Language, and presented in Appendix C. The implementation details of the program are described below.

One of the key issues in a simulation is to map the physical time to the simulation time within the program. This parameter indicates the total simulation time for which the trace has been generated. A global clock forms the simulation time, and it maps to each unit of time spent in the network. Initialization of all node parameters, after reading input values is performed first. The periods of all messages are then tested. Each message has a priority assigned to it depending on the first parameter identification character PID contained in the message.

The simulation starts with the arrival of a new message. The first arrival of a message is determined by its release time. Once a periodic message is released it arrives at the specified periodic rate. The completed transfer of a message is followed by an idle state of 10 bit times. If no message arrives, then the bus is in an idle state. Each idle state results in the incrementing of an idle time counter. When one or more messages are generated at the same point in time, a message cycle is started. If more than one message arrives, then there is a conflict and collision is detected. The eventual transmitter of the message starts the message transfer. A conceptual flow diagram of the bit transfer, is shown in Figure 3. Every bit put onto the bus is a logical value, found by testing the appropriate bit position within the transmitted message using Boolean logic. Every bit transfer results in an additional bit time being spent in the simulation. After the lapse of a bit time the bit is received by all receivers simultaneously.

Errors are generated at random times using a uniform random number generator. When an error occurs, the checksum of the transmitted message will not tally with the original checksum of the message causing a checksum error. The transmitter detects bit errors, while the receiver detects the checksum error.

CHAPTER V

ANALYSIS AND RESULTS

5.1 STATISTICAL AND GRAPHICAL ANALYSIS

The input data for the simulation consists of a set of messages with some additional data to facilitate testing, and to obtain various network performance measures. The input data format that is used in the input data file is as follows:

```
Run-time Output Flag
Seed
Simulation Time (bit times)
Network Speed (bits/sec)
t1 (bit times) - Sampling Start Time
t2 (bit times) - Sampling End Time
Error Rate
Number of Nodes
        Node Id
        Number of Messages
                Release time (bit times)
                Deadline (bit times)
                Type of message (periodic or sporadic)
                Period or Arrival Rate
                MID
                Number of Parameters
                        PID     Data
```

Simulation runs have been performed in the time range of 2000 bit times to 20,000 bit times. This is done to accommodate for load variations, error rates, and irregularity in message generation times. The Run-time Output Flag is used to indicate if an output file should be created to store run-time output. The Run-time output file may be very large. Its size, in bytes, is roughly equal to the simulation time (measured in bit time). The Seed is a number that seeds the random number generator that is used for the error generation and the arrival rates of sporadic messages. The Simulation Time is the length of time that the simulation runs (measured in bit times). The Network Speed is provided to convert bit times to other units such as seconds and milliseconds. The t1 and t2 parameters are the sampling times when the statistics are taken. The value of t1 could be zero or any other value before the termination of the simulation and t2 is generally equal to the simulation duration. The error rate parameter is used as a means of varying error generation points and rates. Each node has a node id associated with it, which starts from one up to the total number of nodes. Each node is defined with a certain number of messages, which include the MID, PID and data. The Checksum for each message is calculated at the end of each message. Any node can generate only one message at a time. So, an upper bound on message arrivals is the number of nodes in the network. Each message in turn has a period if the message is periodic, or an arrival rate if the message is sporadic, a release time, and a deadline. The units are all in bit times.

Verification of the simulation is performed on a single node with a single periodic message, having a period of 100 bit times and running at a Network Speed of 9600 bits/sec. A run of 1000 bit times, with no errors and no collisions, produces the following results:

Total number of messages transmitted = 10

Idle time = 62.5 msec	Busy time = 41.67 msec
Network Load = 40.04 %	Network Throughput = 3843.844 bits/sec

Average Response Time = 1.04 msec

It is obvious that a message with a period of 100 bit times arrives 10 times in a run of 1000 bit times, hence transmitting 10 messages. The sum of the idle and busy times gives the total simulation time. The slight variation in the results is due to the conversion of the bit time to seconds. The duration of a single message on a single node can also be calculated. The single message consists of a single MID comprised of 10 bits with the start bit, eight bits of MID and then the stop bit. It is then followed by a PID again consisting of 10 bits which has only 1 data character that also consists of 10 bits. The message is terminated by a checksum character of 10 bits. Hence the total message consists of 40 bits. Each message is preceded by an idle bus time of 10 bits. Hence the network busy time is 40 times 10 which is 400 bit times which, when converted to seconds, is 41.67 msec; i.e., 400/9600. The network load is the ratio of the busy time to the total time which can be evaluated as 400/1000 which is 40%. The network throughput is the ratio of the data time to the total time. Since there are no errors during this simulation, the data time will be the same as the busy time which results in a throughput of 3840 bits/sec.

The simulation is performed on 7 different input message sets, with 2, 5, 10, 20, 30, 40, 50 nodes each. Each node transmits a set of messages. These input conditions are labeled as 1, 2, 3, 4, 5, 6, and 7 respectively for the curves and load points in the graph. This gives a variation in the offered load on the network. The input files for the 7 different input message sets are given in Appendix D. The statistics are computed and the output is analyzed at 5 sampling intervals. The statistics at the end of a run of 50 nodes for a Simulation Time of 10,000 bit times is presented in Appendix E. The output includes network and node statistics that help in making a comprehensive performance evaluation of the network. The outputs for the corresponding input of 50 nodes is given in Appendix F. Due to the stochastic nature of the simulation model, the observed performance of the system is only an estimate of the true performance. The steady state behavior of the simulation is therefore analyzed. A steady state simulation is a simulation whose objective is to study long-run or steady state behavior of a nonterminating system [BANK84]. The following discussion analyzes the results obtained out of the statistics using some representative graphs.

Network Load is defined as the ratio of the utilized bus time to the total bus time; that is, Network Load = Busy_time / Total_time. The utilized time includes useful message transmission. The five different runs of seven different inputs produce graphs as show in Figure 4.

Initially, all runs have a high load signifying the simultaneous release of messages by all nodes on the network. Then, there is a near exponential decrease in the load as the distribution of message generation times is more varied. In the end, the graphs tend to become horizontal lines representing a more steady state system behavior. It can be observed that as the message sets become larger the exponential decrease becomes smaller. This signifies that as the message sets increase in size, more transmissions are clustered together.

Ideally, the network load will not exceed 30-40%, hence another sample run of all nodes is taken, although this time the simulation time is higher, starting from approximately one second to two seconds. This helps in analyzing the load variants of the network with 30 nodes or more. Keeping the number of messages transmitted fixed, for a particular input set, we can see that from an initial high load, the load keeps decreasing and almost becomes a horizontal line as time increases. This is due to the fact that the busy time becomes almost a constant. This can be clearly seen for 2, 5 and 10 nodes in Figure 5.

Network Throughput is defined as the total number of bits transmitted per second, and is given by the formula, Network throughput = Data_time / Total Time, where Data_time is the time when the messages are successfully transmitted on the bus. Network Throughput versus Network Load is shown in Figure 6. A predicted behavior is seen in the form of a linear or logarithmic graph, with a gradual rise as the Network Load increases. This is attributed to the fact that more messages are transmitted as the load is increased.

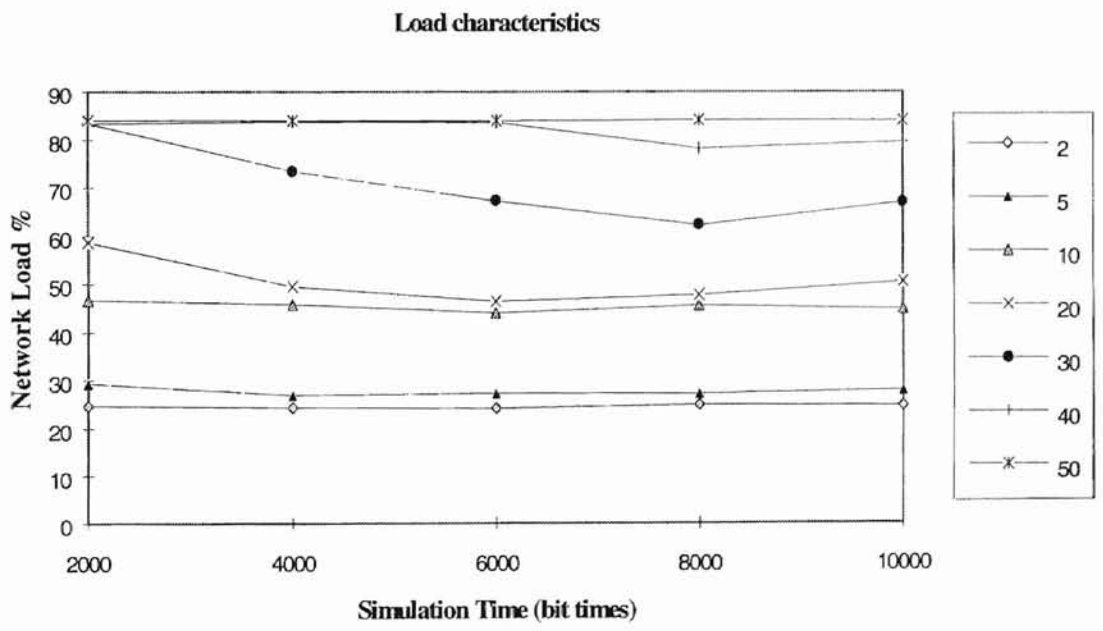

Figure 4: Network Load Characteristics

The *time analysis* graphs of Figure 7 shows the two major time parameters analyzed in the simulation with respect to the load. It can be seen that the sum of the busy and idle times is equal to the simulation time. Idle time is the time at which no transmissions take place; i.e., the bus is in an idle state.

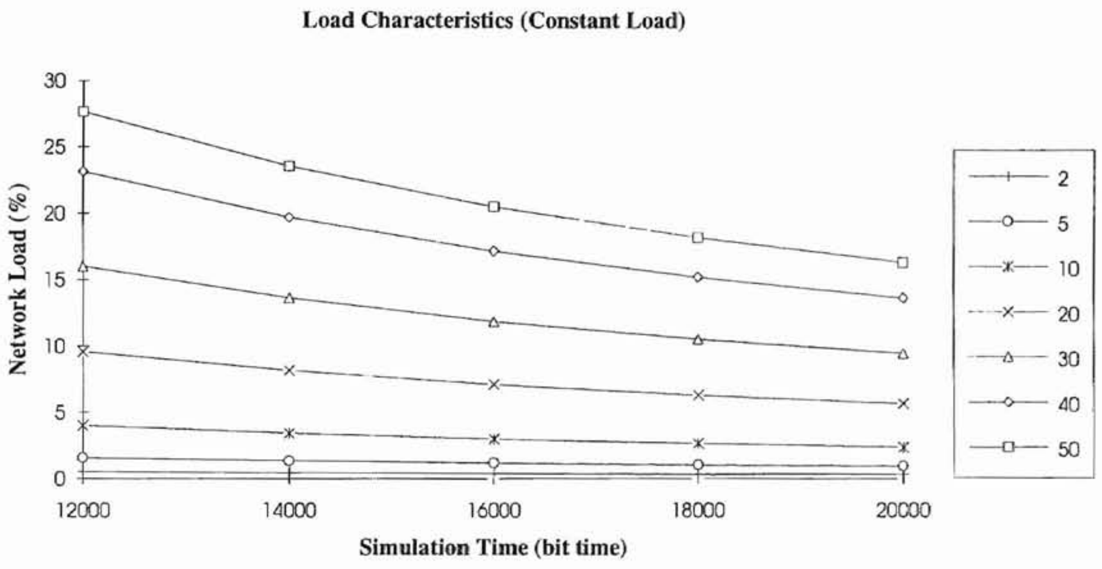

Figure 5: Network Load Characteristics for a constant load

The idle time graph decreases linearly with increased load. This is obvious from the fact that as more messages are generated, the bus is free for smaller amounts of time. The busy time is the time when the bus is busy due to transmission of messages. The busy time graph increases linearly with increased load. This is obvious from the fact that as more messages are generated the bus is busy most of the time.

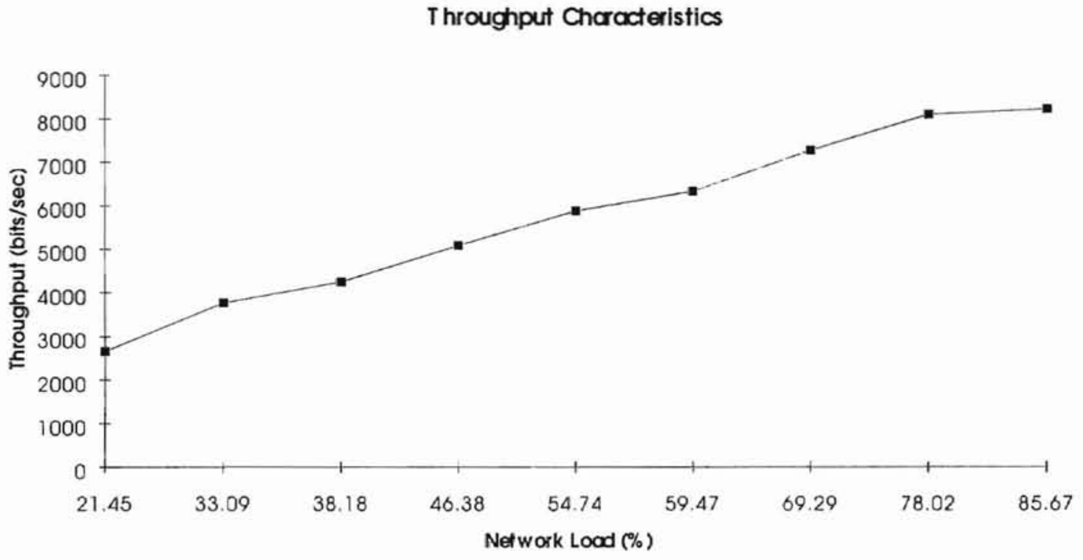

Figure 6: Network Throughput Characteristics

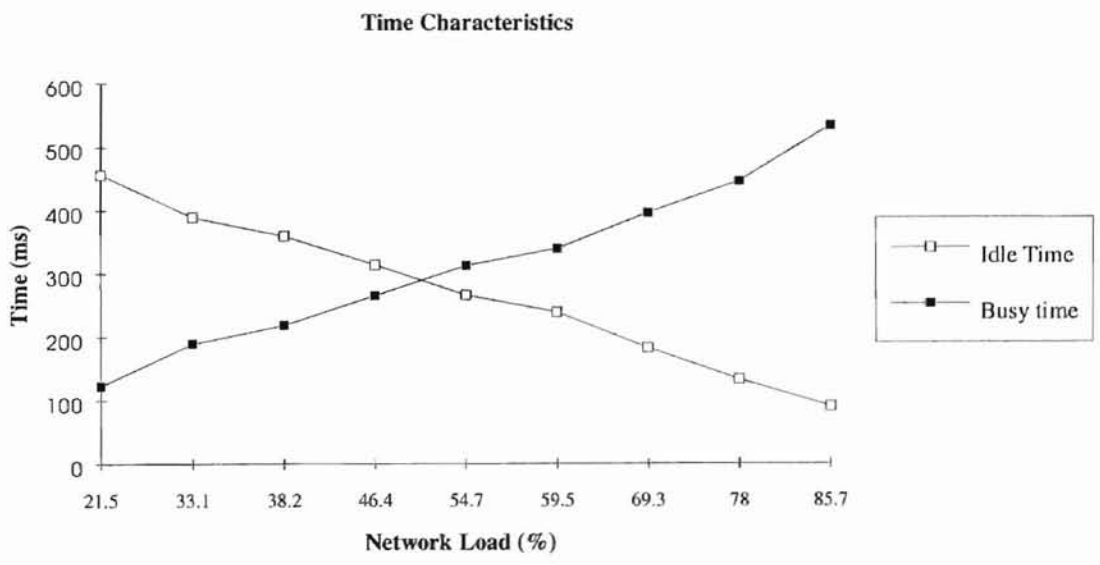

Figure 7: Busy and Idle Time Characteristics

The *Average Response Time* in a network is the average amount of time taken by all messages to gain bus access, once they are generated. The graphs in Figure 8 show the changes in the average response times at various sampling points within the simulation trace. The graphs for the eight topological conditions present interesting characteristics of the J1587 Standard. The first two load conditions have negligible amount of average response times. Fewer nodes offer a lower load and produce a more stable average response time, while message sets with more than 20 nodes offer greater loads and exhibit a sharp increase in the response times.

Keeping a constant network load, we can also see that the average response time increases as the number of nodes increase. The simulation statistics are gathered from one second to two seconds to determine a steady state behavior as shown in Figure 9.

Another parameter under study in the network is missed deadlines. This graph shows the typical characteristics of messages ranging from high priority to low priority. This is taken for a given input sequence of nodes and messages. Figure 10 traces the missed deadline characteristics. As expected, lower loads produce fewer missed deadlines, while larger loads have more missed deadlines. The graph is horizontal in the beginning up to approximately 46% and then gradually changes until it reaches approximately 80%, where the slope becomes extremely steep due to a large number of messages missing their deadlines. We can see that higher priority messages do not miss as many deadlines as the lower priority messages.

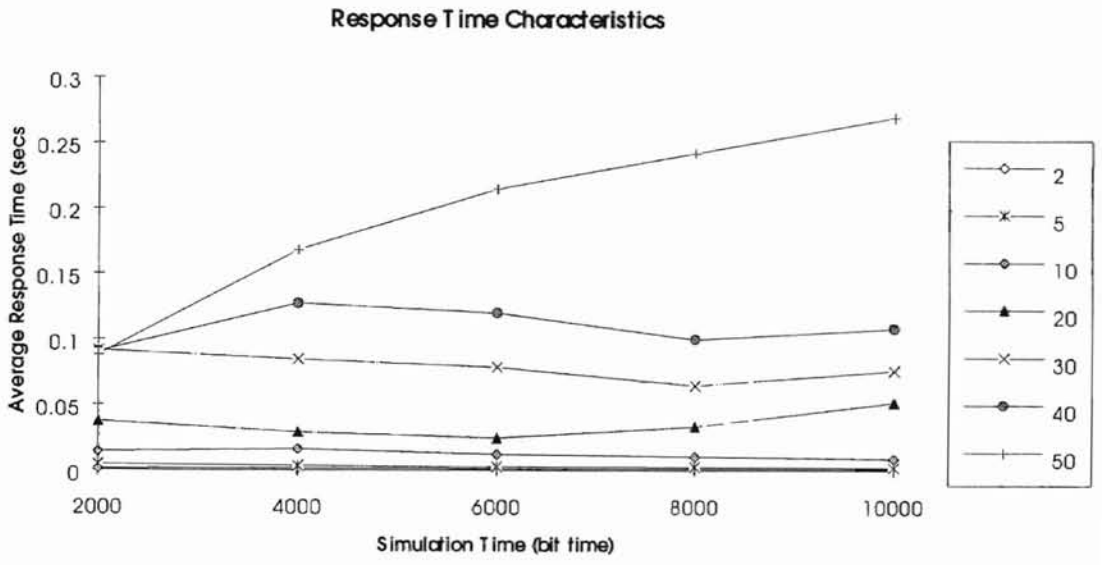

Figure 8: Average Response Time Characteristics

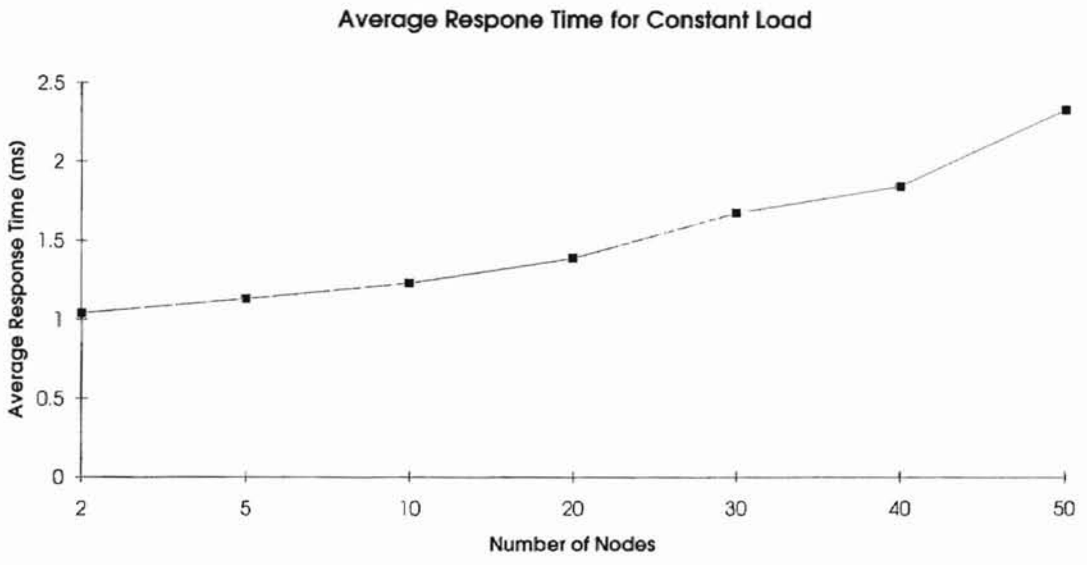

Figure 9: Response Time Characteristics with a Constant Load

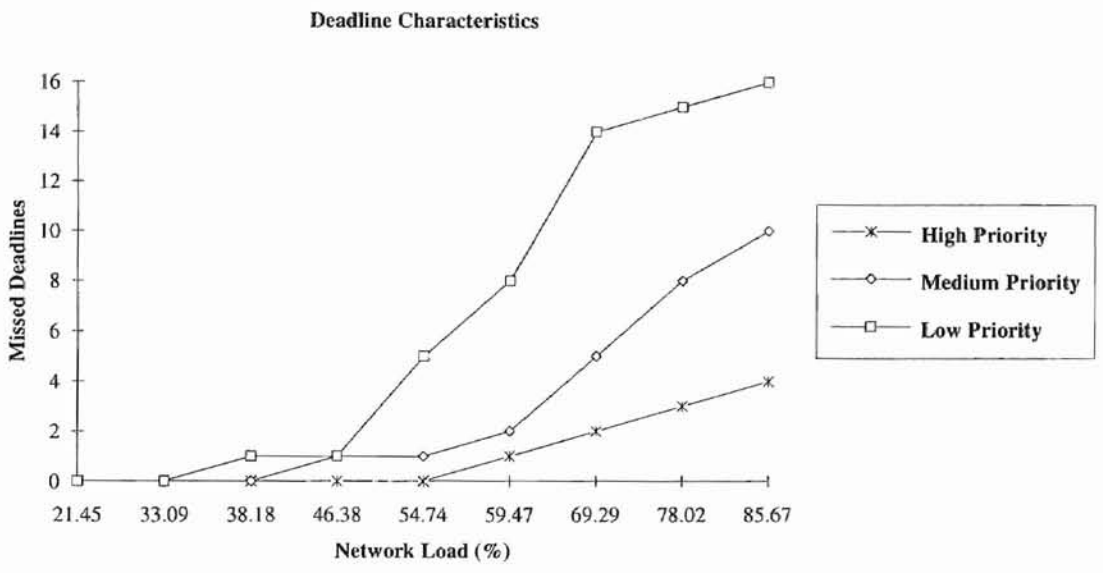

Figure 10: Missed Deadlines Characteristics

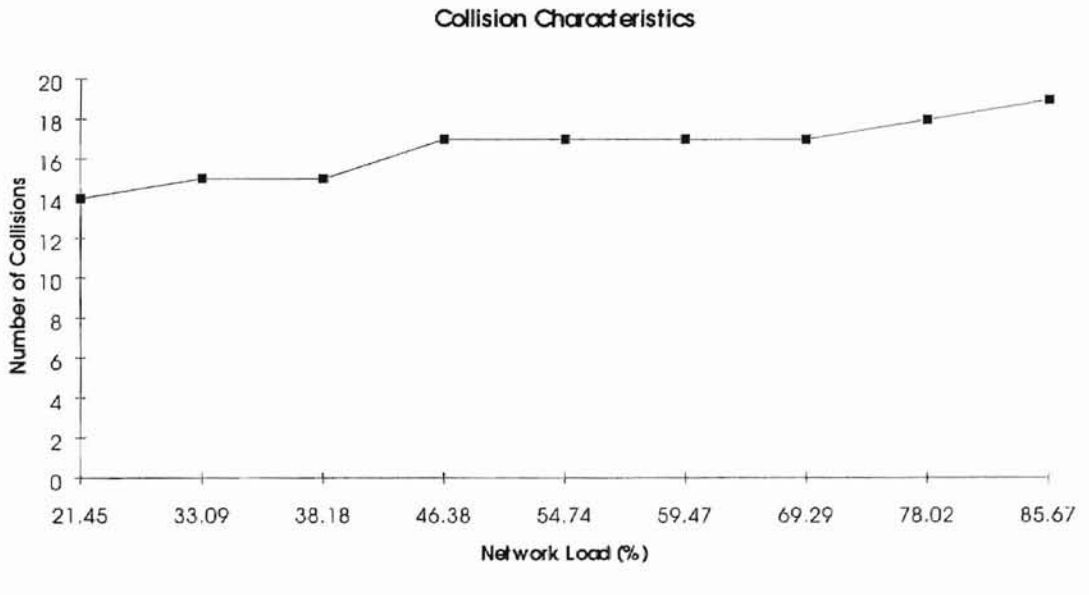

Figure 11: Collision Characteristics

NODE	*No. of messages*	*Priority*	*Release (bit time)*	*Deadline (bit time)*	*Period (bit time)*	*No. of bits per message*
1	1	6	100	200	17999	140
2	1	2	100	200	11990	60
3	1	3	410	500	18980	40
4	2	7	410	450	7990	80
		8	30	38	18990	100
5	1	1	1300	1700	17999	40
6	1	5	400	450	21999	40
7	1	2	450	900	11999	40
8	1	7	450	490	28850	80
9	1	6	1900	2000	21900	120
10	1	2	2200	2895	29500	40

Table 2: Input Node Characteristics

The graph of Figure 11 shows the number of collisions at different load points. This is a linear graph that grows gradually. This analysis is only for a given input set. But the general behavior for any message set is the same, as the load increases the number of collisions also increases.

Another interesting feature that can be observed is the individual node behavior for a given input set of nodes. Each node behavior depends on its input parameters such as the release time, the period and the deadline. Depending on these parameters, there might be collisions and missed deadlines. These can be observed from the Table 2 and 3, which provide the input parameters of 10 nodes and their corresponding output values.

Node	*No. of msgs sent*	*No. of collisions*	*No. of missed deadlines*	*Average Response Time*
1	2	0	1	0.00448
2	2	1	0	0.00104
3	2	0	0	0.01604
4	5	2	3	0.01304
5	2	0	0	0.00104
6	1	0	1	0.00708
7	2	0	0	0.02484
8	1	1	1	0.05292
9	1	0	1	0.00104
10	1	0	0	0.00104

Table 3: Output Node Characteristics

For an input set of 10 nodes and 11 messages, the total number of messages transmitted during a simulation run of 2.03125 seconds was 19. Some of the other characteristics are listed below:

Idle time = 1.94510 sec   Busy time = 0.09177 sec

Network Load = 4.5199 % Network Throughput = 649.8462 bits/sec

Average Response Time = 0.001169 sec

The given input set, with each node having one or two message sets, is as shown in Table 2. Each message has a corresponding release time, deadline, and period or arrival rate depending on whether it is periodic or sporadic. From Table 3 we can see that each node has a behavior with respect to the average behavior. Some messages have more collisions than the others, and some miss deadlines while others do not.

CHAPTER VI

CONCLUSIONS

6.1 SUMMARY OF RESULTS

The J1587 protocol has been analyzed for load conditions varying from 0 to 100%. Most results show good performance for loads under 50%. The inclusion of errors in the simulation produce measures of error tolerance. A topology with 50 nodes results in more messages missing their deadlines.

On the whole, the network performs admirably with a load as high as 50%. The throughput achieved for this scenario is around 5000 bits/sec for a network with a capacity of 9600 bits/sec. From the time characteristics it is clear that there is a large idle time that can provide for some additional loading if necessary. Response times are also faster for loads under 50%.

6.2 CONCLUSIONS

From the analysis, it is clear that the network begins to degrade under severe loads. Since no message can afford to miss its deadline, guaranteed performance is mandatory. Considering the fact that a pessimistic approach is chosen in terms of release times, a better performance can be assured for a more typical J1587 environment.

The backoff and retransmission protocol used to resolve bus contention seems to work well with the message set. It provides for no more than two consecutive collisions per node, thus enhancing the performance by avoiding delays. Overall, the J1587 has proven to be an efficient real-time network.

6.3 FUTURE RESEARCH

Since J1587 is a developing protocol, research findings are essential for its growth. Every network goes through an evolution, and its success depends on how well a performance evaluation is made before formally setting standards. This thesis opens up several avenues for research. Studies in the changes of key network parameters have revealed some network limitations. The simulation program has been crucial in arriving at the above conclusions. The bit by bit logic within the simulation has aided in error analysis. It is also useful for making design changes in message formats. A full-scale investigation of different topologies could be made. This could give insight into a wide range of design issues. One of the aspects left out of the study was fault tolerance. Although the fault confinement logic has been implemented within the simulation, it could not provide adequate results. This was due to the fact that not enough errors were generated to create many faulty messages. Modifying the simulation could give some fault tolerance measures. Additionally, testing of the bus with asymmetric loads has not been done in this thesis.

REFERENCES

[ARNE97] Arnett, J.D., *A high performance solution for in-vehicle networking Controller Area Network(CAN)*, Earthmoving Industry Conference, Peoria, IL, April 7, 1987, SAE paper #870823.

[BOSC91] Robert Bosch GmbH, *CAN Version 2.0b,* Stuttgart, Germany, September, 1991.

[BANK84] Banks, J., and Carson, J.S. *Discrete-Event Simulation*, Prentice-Hall Publishing, Englewood Cliffs, NJ.

[BRAT87] Bratley, P., Fox, B.L. and Schrage, L.E., *A Guide to Simulation*, Springer-Verlay Publishing, New York.

[EYHO89] Ey, H., *Controller Area Network (CAN) Components*, Electronic Component and Application, Vol. 9, No. 3, 1989, pp. 155-158.

[GREL91] Grela-M'Poko, B., Ali M.M., and Hayes, J.F., *Approximate analysis of asymmetric single-service prioritized token passing systems*, IEEE Transactions on Communications, Vol. 39, No. 7, pp. 1037-1040.

[IBE89] Ibe, O.C., X. Cheng, (1989). *Approximate analysis of asymmetric single-service token-passing systems*, IEEE Transactions on Communications, Vol. 37, No. 6, pp. 572-577.

[JEAN94] Tsao-Jean Leu, *Performance Analysis of a Controller Area Network subject to Asymmetric Traffic Loads,* Masters Thesis, Department of Computer Science, Oklahoma State University, 1994.

[JEFF91] Jeffay, K., et. al., *On non-preemptive scheduling of periodic and sporadic tasks*, Proceedings of the twelfth IEEE real time systems symposium, San Antonio, Texas, Dec. 1991, pp. 129-139.

[JEFF92] Jeffay, K., *Scheduling sporadic tasks with shared resources in hard real time systems*, Proceedings of the thirteenth IEEE real time systems symposium, Phoenix, Arizona, Dec. 1992, pp. 89-99.

[JOHN86] Johnson, W. and Volk, J., *A proposal for a vehicle network protocol standard*, Feb. 1986, SAE paper # 860392.

[KIEN86] Kiencke, U., Dais, S. and Litschel, M., *Automotive serial controller area network*, International Congress and Exposition, Detroit, Michigan, Feb. 24-29, 1986, SAE Paper # 860391.

[LAW91] Law, A.M., and Kelton, W.D. (1991). *Simulation Modeling and Analysis*, McGraw-Hill Publishing, New York.

[MARK93] Mark R. Stepper, *J1939 High speed serial communications - the next generation network for heavy duty vehicles,* Society of Automotive Engineers, May 1993, SAE Paper # 931809.

[MIES86] Miesterfeld, F.O.R, *Chrysler Collision Detection (C^2D) - a revolutionary vehicle network,* International Congress and Exposition, Detroit, Michigan, Feb. 24-28, 1986, SAE Paper # 860389.

[NATA93] Pennathur N., *Bitwise Simulation of Controller Area Network,* Masters Thesis, Department of Computer Science, Oklahoma State University, 1993.

[PAUL92] Paul, F., *Simpack: Getting Started with Simulation Programming in C and C++,* ACM Transactions on Modeling and Computer Simulation, 1992.

[PHAI86] Phail, F.H., Arnett D.J., *In-vehicle networking - serial communications requirement and directions,* SAE Paper # 860390.

[PHAI88] Phail, F.H., *Controller area network - an in-vehicle network solution,* International Summer Meeting of the ASAE, Rapid City, SD, Paper #883021.

[SAE87] *Serial data communications between microcomputer systems in heavy duty vehicle applications.*, June 1987.

[SAE88] *Recommended practice for electronic data interchange between micro-computer systems in heavy-duty vehicle applications.*, January 1988.

[SENI88] Senior, J.M., Higginbottom, G.N. and Ryley, A.N., (1988). *Industrial LANs: access protocol performance and traffic load/throughput investigation*, Proceedings of the International Conference on Network Technology and Architectures, London, U.K., pp. 1191-1193.

[STEP93] Stepper, M.R., (1993). *J1939 High speed serial communications, the next generation network for heavy-duty vehicles*, SAE Paper #931809.

[TAKI88] Takine, T.,Takahashi, Y., and Hasegawa, T., (1988). *Exact analysis of asymmetric polling systems with single buffers*, IEEE Transactions on Communications, Vol. 36, No. 10, pp. 1119-1127.

[XU93] Xu,J. and Parnas, D.L., *On satisfying timing constraints in hard real time* *systems*, IEEE transactions on software engineering, Vol. 19, No. 1, Jan. 1993, pp. 70-84.

[WANG91] Wang, Z., *Analysis and Design of Controller Area Networks*, MS Thesis, Oklahoma State University, December 1991.

[WANG92] Wang, Z., Stone, M. and Lu, H., *A message priority assignment algorithm for CAN based networks*, Proceedings of the twentieth Annual Computer Science Conference, Mar. 1992, pp. 25-32.

APPENDIX A

GLOSSARY

BAUD:	The maximum number of analog signal transitions per second that can occur on a channel. In this coding system, it is the reciprocal of the bit time.
BIT TIME:	Duration or period of one unit of information.
CHARACTER TIME:	The duration of one character. The character must start with a low-logic bit, then 8 bits of data followed by a high-logic level stop bit.
COLLISION:	It is the overlapping transmissions when two or more nodes try to attempt to transmit messages simultaneously.
COLLISION DETECTION:	It is the ability of a transmitting node to detect simultaneous transmission attempts on a shared medium.
CONTENTION:	A state of the bus in which two or more transmitters are turned on simultaneously to conflicting logic states.
IDLE LINE:	The condition that exists when the bus has remained in a continuous high-logic state for at least 10 bit times after the end of the last stop bit.
IDLE STATE:	The state that produces a high-logic level on the input of the bus receiver when all transmitters on the network are turned off.
NODE:	A receiver or transmitter circuit connected to the bus.
PROTOCOL:	It is a formal set of rules and regulations governing the format and relative timing of message exchange between two communication systems.
START BIT:	Initial element of a character defined as a low-logic

	level of 1 bit time duration as viewed at the output of the bus receiver.
STOP BIT:	Final element of a character defined as a high-logic level of 1 bit time duration as viewed at the output of the bus receiver.

APPENDIX B

SPECIFICATIONS FOR J1587 SIMULATOR SYNTAX

<input>	:=	<simulation time><network speed>
		<input messages><bandwidth>
<output>	:=	<runtime output><summary statistics>

Semantics

1. The simulator accepts the <input>, processes it and produces the <output>

2. Output includes the runtime output and the summary statistics.

INPUT

SYNTAX

<simulation time>	:=	<long>
<network speed>	:=	<int>
<input messages>	:=	<input message><input messages> |
		<input message>
<input message>	:=	<MID><parameters><checksum>
<MID>	:=	<character> [SAE89]
<character>	:=	<start_bit><byte><stop_bit>
<start_bit>	:=	<0>

<byte>	:=	<0-255>
<stop_bit>	:=	<1>
<parameters>	:=	<parameter><parameters> |
		<parameter>
<parameter>	:=	<PID><data>
<PID>	:=	<character> [SAE89]
<data>	:=	<character> | <character>^{2}
	:=	<character>^{4} | <character>^{8}
<checksum>	:=	<character>

Semantics

1. MID for any transmitter in the system is unique.

2. PID range from 0 to 255, mainly comprising of the following types:

   a. Single data character (0-127)

   b. Double data characters (128-191)

   c. More than two data characters (192-253)

   d. Data link escape PID (254)

   e. Extension PID (255)

3. The parameter data types could be one or more of the following types:

   a. Binary Bit-Mapped (B/BM) (1 char)

b. Unsigned Short Integer (Uns/SI) (1 char)

c. Signed Short Integer (S/SI) (1 char)

d. Unsigned Integer (Uns/I) (2 char)

e. Signed Integer (S/I) (2 char)

f. Unsigned Long Integer (Uns/LI) (4 char)

g. Signed Long Integer (S/LI) (4 char)

h. Alphanumeric (ALPHA) (1 char)

i. Single-Precision Floating-Point (SP/FP) (4 char)

j. Double-Precision Floating-Point (DP/FP) (8 char)

4. Each parameter is composed of the parameter data length, the data type, resolution, maximum range, transmission update period, and the message priority.

5. All messages will have a priority assigned in the range of 1 to 8. The most critical message has a priority of one. {Refer Table 1-J1708}

6. Message length does not exceed 21 characters.

7. SIDs are numbers ranging from 0-255 for each controller or MID.

8. Failure Mode Identifier (within the data) describes the type of failure detected in the subsystem identified by the PID or SID.

9. The range of the "int" is from -32768 to 32767.

10. The range of the "long" is from -2147483648 to 2147483647.

11. The simulation time is in bit times which is $104.17\ \mu s \pm 0.5\%$.

12. The network speed is 9600 bits per second.

13. The idle line is the condition that exists when the bus has remained in a continuous

high logic state for atleast 10 bit times after the end of the last stop bit.

14. Bus Access time is a time duration equal to the minimum time of an idle line plus the product of 2 bit times and the message priority.

    The relationship can be expressed as follows:

    $$T_a = T_i + [2*T_b] * P$$

    where:

    T_a = bus access time

    T_b = bit time, or period of one unit of information

    P = Message Priority

    T_i = Minimum time duration of an idle line.

15. The bus can support a minimum of 20 standard nodes.

16. A character consists of 10 bit times. The first bit called the start bit is <0> and the tenth (last) bit called the stop bit is <1>.

17. A message is always preceded by an idle state of duration equal to or greater than the appropriate bus access time. The length of time between characters within a message will not exceed 2 bit times.

18. The transmitters monitor the message identification portion of their message to determine if another transmitter has attempted to gain access to the bus at the same time. If collision is detected, the transmitter relinquishes control of the bus after completing the transmission of the current character. Each transmitter now waits their predefined bus access time. If the transmitter experiences two consecutive crashes then the bus access time is calculated as follows [SAE89]:

$$T_a = T_i + [2*T_b]*[P_2 + 1]$$

where:

T_a = bus access time

T_i = minimum time duration of an idle line

T_b = Bit time, or period of one unit of duration

P_2 = A three-bit pseudo random number

OUTPUT

Syntax

<output>	:=	<runtime output><summary statistics>
<runtime output>	:=	<transmissions><errors>
<summary statistics>	:=	<total time><summary of each node>
		<summary of all nodes>
<total time>	:=	<end sampling time>-
		<start sampling time>
<summary of each node>	:=	<node id><performance measures>
<summary of all nodes>	:=	<network load><network throughput>
		<average response time>
		<number of missed deadlines>
		<number of errors>
		<number of collisions>

SEMANTICS

1. Network load is the ratio of the utilized bus time to the total time, i.e. the percentage of time the network was in use.
2. Network throughput is the total number of valid bits transmitted per second.
3. Types of errors could be form errors and checksum errors.

APPENDIX C

SIMULATION PROGRAM

```
/*=== FILE: const.h ==========================================*/
/*
   Contents
   --------
   J1587 simulator constants

*/
/*=========================================================*/

#define NODE_MSGS             115
#define TOT_MSGS              460
#define MAX_NODES             60
#define MAX_NAME              10
#define MAX_MSG_LEN           132
#define MAX_NO_OF_PID         24
#define MAX_NO_OF_DATA        8
#define YES                   1
#define NO                    0
#define BUSY                  1
#define IDLE                  0
#define OVER                  0
#define FAILURE               0
#define SUCCESS               1
#define BIT_ERROR             2
#define FORM_ERROR            5
#define PERIODIC              0
#define SPORADIC              1
#define ACTIVE                0
#define PASSIVE               1
#define BUS_OFF               2
#define A                     16807L      /* multiplier (7**5) for 'ranf'   */
#define M                     2147483647L /* modulus (2**31-1) for 'ranf' */

/*=========================================================*/
/*=== END FILE: const.h =====================================*/
/*=== FILE: defs.h=========================================*/
/*
  Contents
  --------
  J1587 simulator type definitions
```

```
*/
/*=========================================================*/

typedef struct {  /* formatted packet */
        unsigned char start_bit;
        unsigned char stop_bit;
        unsigned char MID;
        unsigned char PID[MAX_NO_OF_PID];
        int total_no_of_parameters;
        unsigned char DATA[MAX_NO_OF_PID][MAX_NO_OF_DATA];
        int tno_of_data[MAX_NO_OF_PID];
        unsigned char CHECKSUM;
} FRAME;

typedef struct
{
 char msg_name[20];
 long release;
 long deadline;
 char msg_id[30];
 int msg_type;
 int coll_count;
 long period;
 float arrival_rate;
 int error_flag;
 int transfer;
} MESSAGE;

typedef struct {
 MESSAGE msg[NODE_MSGS];
 FRAME frame;
 FRAME fram[MAX_NODES];
 int total_msgs, tx_msgs, curr_msg;
 int transfer, bit_val;
 unsigned char err_flag, err_delim;
 int  status;
 int  error_msgs;
 int  trans_err_cnt; /* used for node status */
 int  recv_err_cnt;
 long response_time;
 long turnaround_time;
 int  collisions;
 int  missed_msgs;
 char node_name[MAX_NAME];
} NODE;

/*==========================================================*/
/*=== END FILE: defs.h =======================================*/
/*=== FILE: global.h===========================================*/
/*
   Contents
   --------
   J1587 simulator global variables

*/
/*==========================================================*/

int runtimeOutput;           /* Flag - display runtime output = 1            */
long simDuration;            /* Simulation time (in bit time)                */
long networkSpeed;           /* Channel capacity (in bps)                    */
int seed;                    /* Random number seed                           */
long t1, t2;                 /* Start, end sampling points (in bit time)     */
float errorRate;             /* Rate of error generation (in errors/bit time) */
int totalNodes;              /* Total number of devices (nodes)              */
NODE node[MAX_NODES];  /* Node information                                   */
NODE receiver;               /* Receiver node information                    */
int ones;                    /* Number of recessive bits in a sequence       */
int zeros;                   /* Number of dominant bits in a sequence        */
int count;                   /* Number of messages that have arrived for tx  */
int prv_bit;                 /* Previous bit tx on the bus                   */

long idle_time;              /* Bus idle time (in bit times)                 */
long busy_time;              /* Bus busy time (in bit times)                 */
long response_time;          /* Total frame response time (in bit times)     */
long turnaround_time;        /* Total frame turnaround time (in bit times)   */
int collisions;              /* Total number of collisions (arbitrations)    */
int missed_msgs;             /* Total number of frames missing deadline      */
int msg_time;                /* Time required to tx a frame                  */
long data_time;              /* Total time spent tx data or remote frames    */
int error_msgs;              /* Total number of error frames tx              */
int tx_msgs;                 /* Total number of frames tx                    */
int retx_msgs;               /* Total number of frames retx                  */
int pos;                      /* Position within frame being tx              */
long tic;                    /* Logical clock (in bit time)                  */

int bit_count;               /* Total number of bit errors                   */
int form_count;              /* Total number of form errors                  */

int bus_value;               /* Current bus value (0 or 1)                   */
int  bus_flag;                /* Bus flag, IDLE, BUSY, etc.                    */
int  m,n;                     /* Message number, node number                   */

long In[] = {0L,              /* seeds for streams 1 thru 15                   */
  1973272912L, 747177549L,  20464843L, 640830765L, 1098742207L,
   78126602L,  84743774L, 831312807L, 124667236L, 1172177002L,
  1124933064L, 1223960546L, 1878892440L, 1449793615L,  553303732L};

int strm;                     /* index of current stream for ranf()            */

/*=======================================================*/
/*=== END FILE: global.h=================================*/

/*=== FILE: protos.h=====================================*/
/*
   Contents
   --------
   J1587 simulator prototypes

*/
/*=======================================================*/

void get_input_parm(void);
void sys_init(void);
void read_file(char *);
void msg_cycl(void);
void statistics(void);

void msg_transfer(int);
int transmit();
void receiver_frame_init(void);
int msg_filter(int, int, int, int, int, int);
int receive(int, int, int, int, int, int);
int get_bit(unsigned char, int);

double ranf(void);
double uniform(double,double);
void rand_init(int);
int rand_rate(float);

void inc_tic(int);
void stopsim(void);
int pread_file(int sent_pid);
/*=======================================================*/
/*=== END FILE: protos.h ================================*/
```

```
/*====FILE: sim.c =============================================*/

/*
   Contents
   --------
   J1587 Simulator

   Routines in file
   ----------------
   get_input_parm - read input parameters from a file
   sys_init - initialize simulation parameters
   msg_cycl - execute simulation
      msg_transfer - simulate message transfer
         transmit - transmit data or remote frame
            msg_filter - filter received frames
               get_bit - receive a bit
                  rand_rate - uniform random number generator
               receive - simulate reception of data
               inc_tic - increment the clock
   statistics - compute statistics

   Notes
   -----
   Last modified 04/21/95

*/
/*=============================================================*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>
#include <malloc.h>
#include "const.h"
#include "defs.h"
#include "global.h"
#include "protos.h"

FILE *fp_in, *fp_out, *fp_stat, *pin;

/*=============================================================*/
/* This is the driver for the whole simulation. It first calls the        */
/* function to initialize all the simulation parameters, then it calls         */
/* the function to get all the input parameters from the file. Then it         */
/* calls the function to start the simulation. It takes the statistics         */
/* and finally terminates the simulation                                       */
/*=========================================================================*/
void main(void)
{

 sys_init();
 get_input_parm();
 msg_cycl();
 statistics();
 stopsim();
}


/*====================================================================*/
/* This function performs initialization of the simulation parameters     */
/*====================================================================*/

void sys_init()
{
 int i, j;

 ones = 0;
 zeros = 0;
 count = 0;
 prv_bit = 1;
 idle_time = 0;
 busy_time = 0;
 response_time = 0;
 turnaround_time = 0;
 collisions = 0;

 missed_msgs = 0;
 msg_time = 0;
 error_msgs = 0;
 tx_msgs = 0;
 retx_msgs = 0;
 pos = 0;
 tic = 0;

 bit_count = 0;

 bus_value = 1;
 bus_flag = IDLE;

 for (i=0; i<MAX_NODES; i++) {
  for (j=0; j<NODE_MSGS; j++) {
   node[i].msg[j].release = 0;
   node[i].msg[j].coll_count = 0;
   node[i].msg[j].deadline = 0;
   node[i].msg[j].period = 0;
   node[i].msg[j].msg_type = 0;
   node[i].msg[j].arrival_rate = (float) 0.0;
  }
  node[i].total_msgs = 0;
  node[i].tx_msgs = 0;
  node[i].curr_msg = 0;
  node[i].transfer = NO;
  node[i].bit_val = 1;
  node[i].status = ACTIVE;
  node[i].error_msgs = 0;
  node[i].recv_err_cnt = 0;
  node[i].response_time = 0;
  node[i].turnaround_time = 0;
  node[i].collisions = 0;
  node[i].missed_msgs = 0;
 }
}

/*================================================*/
/* This function obtains the simulation input parameters from an   */
/* input file such as inp1.dat. Refer to the input specification for   */
/* the input format                                                  */
/*================================================*/
void get_input_parm(void)
{
 int i, j, num;
 char filename[20];
 char input_filename[24];
 char output_filename[24];
 char stat_filename[24];
 char pri_filename[24];
 char *p,buff[80];

 fprintf(stdout,"\nEnter file name (no extension): ");
 fscanf(stdin,"%s",filename);

 sprintf(input_filename,"%s.dat",filename);
```

```
  sprintf(output_filename,"%s.out",filename);
  sprintf(stat_filename,"%s.sts",filename);
  sprintf(pri_filename,"%s.pri",filename);

  if ((fp_in = fopen(input_filename,"r")) == NULL)
  {
    fprintf(stderr,"Unable to open input file");
    exit(-1);
  }
  fscanf(fp_in,"%d",&runtimeOutput);
  fscanf(fp_in,"%d",&seed);

  if (runtimeOutput)
  {
    if ((fp_out = fopen(output_filename,"w+")) == NULL)
    {
      fprintf(stderr,"Unable to open output file");
      exit(-1);
    }
  }

  if ((fp_stat = fopen(stat_filename,"w+")) == NULL)
  {
    fprintf(stderr,"Unable to open statistics file");
    exit(-1);
  }

  if ((pin = fopen(pri_filename,"r")) == NULL)
  {
    fprintf(stderr,"Unable to open priority file");
    exit(-1);
  }

  read_file(input_filename);
  fclose(fp_in);
}

/*==============================================*/
/* This reads all the simulation parameters from the input file       */
/*==============================================*/
void read_file(char *file_name)
{
  char S[80],dat[7],data[7][8];
  int r,b,len,no_of_msgs[MAX_NODES];
  int no_of_parameters[NODE_MSGS],no_of_data[NODE_MSGS][10],data_count;
int msg_count, PID_count, node_count,node_id,data_flag;
unsigned char sum;

b=len=data_flag=0;
msg_count=PID_count=node_id=node_count=data_count=0;
for(r=0;r<MAX_NODES;r++) no_of_msgs[r]=0;
for(r=0;r<NODE_MSGS;r++) no_of_parameters[r]=0;
for(r=0;r<NODE_MSGS;r++)
  for(b=0;b<11;b++) no_of_data[r][b]=0;

/* get the simulation time */
fgets(S,79,fp_in);
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
simDuration = atof(data[0]);

/* get the network speed */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
networkSpeed = atof(data[0]);

/* get the t1 */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
t1 = atof(data[0]);

/* get the t2 */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
t2 = atof(data[0]);

/* get the error rate */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
errorRate = atof(data[0]);
```

```
/* get the number of nodes */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
totalNodes = atoi(data[0]);

fgets(S,79,fp_in);
while(!feof(fp_in) && (node_count != totalNodes)) {
 /* get the node_id */
 len=strlen(S);
 S[len-1] = '\0';
 sscanf(S,"%s",data[0]);
 node_id = atoi(data[0]);

/* get the number of messages */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
node[node_id].total_msgs = atoi(data[0]);

fgets(S,79,fp_in);
while(msg_count != node[node_id].total_msgs) {
 /* get the release time */
 len=strlen(S);
 S[len-1] = '\0';
 sscanf(S,"%s",data[0]);
 node[node_id].msg[msg_count].release = atoi(data[0]);

/* get the deadline */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
node[node_id].msg[msg_count].deadline = atoi(data[0]);

/* get the message type */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
node[node_id].msg[msg_count].msg_type = atoi(data[0]);
```

```
if(node[node_id].msg[msg_count].msg_type == PERIODIC) {
 /* get the period */
 fgets(S,79,fp_in);
 len=strlen(S);
 S[len-1] = '\0';
 sscanf(S,"%s",data[0]);
 node[node_id].msg[msg_count].period = atoi(data[0]);
}
else {
/* get the arrival rate */
 fgets(S,79,fp_in);
 len=strlen(S);
 S[len-1] = '\0';
 sscanf(S,"%s",data[0]);
 node[node_id].msg[msg_count].arrival_rate = atof(data[0]);
}

/* get the MID */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
node[node_id].fram[msg_count].MID = atoi(data[0]);

/* get the number of parameters */
fgets(S,79,fp_in);
len=strlen(S);
S[len-1] = '\0';
sscanf(S,"%s",data[0]);
for(r=0;r<NODE_MSGS;r++) no_of_parameters[r]=0;
no_of_parameters[msg_count] = atoi(data[0]);
node[node_id].fram[msg_count].total_no_of_parameters = atoi(data[0]);
PID_count = 0;

while(PID_count != no_of_parameters[msg_count]) {
      /* get the parameters and the data */
      fgets(S,79,fp_in);
      len=strlen(S);
      S[len-1] = '\0';
      sscanf(S,"%s %s %s %s %s %s %s %s %s",
                dat,data[0],data[1],data[2],
                data[3],data[4],data[5],data[6],data[7],data[8]);
      node[node_id].fram[msg_count].PID[PID_count] = atoi(dat);
      if((node[node_id].fram[msg_count].PID[PID_count] >= 0) &&
        (node[node_id].fram[msg_count].PID[PID_count] <= 127)) {
```

```
        node[node_id].fram[msg_count].tno_of_data[PID_count]= 1;
        no_of_data[PID_count][data_count] = 1;
      }
      else if((node[node_id].fram[msg_count].PID[PID_count] >= 128) &&
              (node[node_id].fram[msg_count].PID[PID_count] <= 191)) {
        node[node_id].fram[msg_count].tno_of_data[PID_count]= 2;
        no_of_data[PID_count][data_count] = 2;
      }
      else {
        data_flag = 1;
        node[node_id].fram[msg_count].tno_of_data[PID_count]= atoi(data[0]);
        no_of_data[PID_count][data_count] = atoi(data[0]);
      }
      for(b=0;b<no_of_data[PID_count][data_count];b++){
        if(data_flag==1) {
         data_flag=0;
         node[node_id].fram[msg_count].DATA[PID_count][b] =
                     atoi(data[b+1]);
        }
        else
         node[node_id].fram[msg_count].DATA[PID_count][b] =
                     atoi(data[b]);
        node[node_id].fram[msg_count].CHECKSUM +=
                     node[node_id].fram[msg_count].DATA[PID_count][b];
      }
      data_count++;
      node[node_id].fram[msg_count].CHECKSUM +=
                     node[node_id].fram[msg_count].PID[PID_count];
      PID_count++;
  }
  node[node_id].fram[msg_count].CHECKSUM +=
                     node[node_id].fram[msg_count].MID;
  sum = node[node_id].fram[msg_count].CHECKSUM;
  node[node_id].fram[msg_count].CHECKSUM = ~sum + 1;
  PID_count=0;
  fgets(S,79,fp_in);
  msg_count++;
  }
  msg_count = 0;
  node_count++;
 }
 node_count=0;
}

/*==========================================*/
```

```
/*
 * New version of the RNG from simpack-2.22, more portable
 * than the original.
 *
 * Author: hrp@cray.com, 94/05/26
 */

/*---- UNIFORM [0, 1] RANDOM NUMBER GENERATOR -----*/
/*                                                  */
/* This implementation should be portable to machines with    */
/* sizeof(int) >=4.                                  */
/*                                                  */
/*--------------------------------------------------------------------------*/
/*=============================================*/
double ranf()
 {
  unsigned int k; long Hi,Lo;
  /* generate product using double precision simulation  (comments */
  /* refer to In's lower 16 bits as "L", its upper 16 bits as "H") */
  Hi = ((In[strm] >> 16) & 0xffff) * A; /* 16807*H->Hi */
  Lo = (In[strm] & 0xffff) * A;        /* 16807*L->Lo */
  Hi += Lo >> 16;              /* add high-order bits of Lo to Hi */
  Lo &= 0xffff; Lo += (Hi & 0x7fff) << 16; /* low-order bits of Hi->Lo */
  k = (int) (Hi >> 15);
  /* form Z + K [- M] (where Z=Lo): presubtract M to avoid overflow */
  Lo-=M; Lo+=k; if (Lo<0) Lo+=M;
  In[strm]=Lo;
  return((double)Lo*4.656612875E-10); /* Lo / (2**31-1) */
 }

/*=======================================================*/
/*------------ UNIFORM [a, b] RANDOM VARIATE GENERATOR -------------*/
/*=======================================================*/
double uniform(a,b)
 double a,b;
  { /* 'uniform' returns a psuedo-random variate from a uniform     */
   /* distribution with lower bound a and upper bound b.        */
   /* if (a>b) then error(0,"uniform Argument Error: a > b"); */
   return(a+(b-a)*ranf());
  }
/*=======================================================*/
```

```
/*=====================================================*/
/* Initializes random number generator based on seed              */
/*=====================================================*/
void rand_init(int seed)
{
 int i;

 strm = seed%15+1;      /* Set stream value 1-15 */
 for (i=1; i<=seed; i++) /* Initialize starting point in the stream */
   ranf();
}

/*=====================================================*/
/*  This function calls the uniform random number generator         */
/*=====================================================*/
int rand_rate(float rate)
{
 int rand_val;

 rand_val = uniform(0.0,1.0) < rate;
 return(rand_val);
}

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

void msg_cycl()
{
 int i, j, k;
 int mode;
 long next;
 double cal_pre,ranno;
 int msg_pending;
 int pri=0;

 bus_flag = IDLE;
 pos = 0;
 rand_init(seed);

 /* cycle until end of the simulation run */
 tic = 0;
```

```
while (tic <= simDuration) {
 count=0;
 next = simDuration;
 msg_pending = NO;
 for (i=1; i<=totalNodes; i++) {
  for (j=0; j<node[i].total_msgs; j++) {
   /* Checking for the type of message whether periodic or sporadic */

   if(node[i].msg[j].msg_type == PERIODIC) {
    if (node[i].msg[j].release < next)  {
     msg_pending = YES;
     k = j;
        n = i;
     next = node[i].msg[j].release;
     }
    else if((node[i].msg[j].release == next) &&
            (node[i].msg[j].release != simDuration)) {
     collisions++;
        node[i].collisions++;
        if(node[i].msg[j].coll_count == 3) {
               pri = rand()%8;
               while(pri >8) pri=rand()%8;
               cal_pre = (2.0 * 1.0 * (pri + 1));
               node[i].msg[j].release = (idle_time + cal_pre) ;
        }
        else {
            pri = pread_file(node[i].fram[j].PID[0]);
               cal_pre = 2.0 * 1.0 * pri;
               node[i].msg[j].release = (idle_time + cal_pre) ;
               node[i].msg[j].coll_count++;
        }
        if(node[n].msg[k].coll_count == 3) {
               pri = rand()%8;
               while(pri >8) pri=rand()%8;
               cal_pre = (2.0 * 1.0 * (pri + 1));
               node[n].msg[k].release = (idle_time + cal_pre) ;
        }
        else {
            pri = pread_file(node[n].fram[k].PID[0]);
               cal_pre = (2.0 * 1.0 * pri);
               node[n].msg[k].release = (idle_time + cal_pre) ;
               node[n].msg[k].coll_count++;
     }
        }
    }
```

```
      else if(node[i].msg[j].msg_type == SPORADIC) {
           ranno = ranf();
           if(node[i].msg[j].arrival_rate <= ranno) {
         msg_pending = YES;
         k = j;
            n = i;
         next = node[i].msg[j].release;
        }
      }
     }
    }
    if ((msg_pending == YES) && (node[n].msg[k].release <= tic)) {
      node[n].transfer = YES;
      node[n].msg[k].transfer = YES;
      node[n].curr_msg = k;
      count++;
    }

  msg_time = 0;
  mode = 1;

  /* resolve bus contention */
  if (count >= 1)
    {
    bus_flag = BUSY;
    if (count != 1)
       collisions++;

    m = node[n].curr_msg;
    count = 0;
    if (node[n].status != BUS_OFF)
      {
      msg_transfer(mode);
      bus_flag = IDLE;
      pos = 0;
      }
    }
  else
    {
    inc_tic(1);
    }
 }
}

/*============================================*/
```

```
/* This function increments the tic and gathers statistics              */
/*==============================================================*/
void inc_tic(int display)
{
 if ((tic>=t1)&&(tic<=t2)) {
  /* gather statistics */
  if (bus_flag==BUSY)
    busy_time++;
  else
    idle_time++;
 }

 if (display) {
  if (runtimeOutput) {
   fprintf(fp_out,"%s", bus_value ? "1" : "0");
   if ((tic+1)%20==0)
     fprintf(fp_out," tic=%ld \n",tic);
   }
  }
  tic++;
  return;
}

/*==================================================================*/
/* This function stops the simulation when tic reaches the simulation   */
/* duration.                                                            */
/*==================================================================*/
void stopsim()
{
 tic=simDuration;
 if (runtimeOutput)
   {
   fprintf(fp_out,"\nStopped.\n");
   fclose(fp_out);
   }
 fclose(fp_stat);
 return;
}

/*==================================================================*/
/* Statistics for the simulation                                        */
/*==================================================================*/

void statistics()
{
```

```
int i;
float Data_time, Total_time;
float Idle_time, Busy_time;
float Response_time, Slack_time, Latency;
float Throughput, Load;

fprintf(fp_stat,"----------------------------------------");
fprintf(fp_stat,"----------------------------------\n");
fprintf(fp_stat,"               Network Statistics\n");
fprintf(fp_stat,"----------------------------------------");
fprintf(fp_stat,"----------------------------------\n");
fprintf(fp_stat,"\nNumber of nodes = %d",totalNodes);
fprintf(fp_stat,"\nNumber of messages transmitted = %d\n",tx_msgs);
fprintf(fp_stat,"\nTotal number of error messages    = %10d", error_msgs);
fprintf(fp_stat,"\nTotal number of collisions        = %10d", collisions);
Idle_time = (float)idle_time/(float)networkSpeed; /* Idle time in sec. */
fprintf(fp_stat,"\nIdle time in the network          = %10.5f sec",
                                   Idle_time);
Busy_time = (float)busy_time/(float)networkSpeed;
fprintf(fp_stat,"\nBusy time in the network          = %10.5f sec",
                                   Busy_time);
if (tx_msgs>0)
 {
 Response_time = ((float)response_time/(float)networkSpeed)/(float)tx_msgs;
 fprintf(fp_stat,"\nAverage response time            = %10.5f sec",
                                   Response_time);
 }
Total_time = (float)(t2-t1)/(float)networkSpeed;
fprintf(fp_stat,"\nSimulation time                   = %10.5f sec",
                                   Total_time);
Load = (Busy_time / Total_time) * (float) 100.0;
fprintf(fp_stat,"\nNetwork load                      = %10.4f %%", Load);
Data_time = (float)data_time/(float)networkSpeed;
Throughput = (Data_time / Total_time) * networkSpeed;
fprintf(fp_stat,"\nNetwork throughput                = %10.4f bits/sec",
                                   Throughput);
fprintf(fp_stat,"\n----------------------------------------");
fprintf(fp_stat,"----------------------------------\n");
fprintf(fp_stat,"               Node Statistics\n");
fprintf(fp_stat,"----------------------------------------");
fprintf(fp_stat,"----------------------------------\n");
fprintf(fp_stat,"Node          No of  No of  No of  No of    ");
fprintf(fp_stat,"Average   \n");
fprintf(fp_stat,"              msgs   error  coll.  deadlines ");
fprintf(fp_stat,"response  \n");
```

```
 fprintf(fp_stat,"            sent  msgs        missed   ");
 fprintf(fp_stat,"time     \n");
 fprintf(fp_stat,"----------------------------------------");
 fprintf(fp_stat,"-----------------------------------");

 for (i = 1;i <= totalNodes;i++) {
  fprintf(fp_stat,"\n%-15d",i);
  fprintf(fp_stat,"%7d",node[i].tx_msgs);
  fprintf(fp_stat,"%7d",node[i].error_msgs);
  fprintf(fp_stat,"%7d",node[i].collisions);
  fprintf(fp_stat,"%7d   ",node[i].missed_msgs);
  if(node[i].tx_msgs !=0)
   Response_time = ((float)node[i].response_time/(float)networkSpeed)/
          (float)node[i].tx_msgs;
  else
   Response_time = 0.0;
  fprintf(fp_stat,"%10.5f",Response_time);
  }
 fprintf(fp_stat,"\n");
}

/*=====================================================*/
/* This function receives a bit sent over the J1587 bus when the node      */
/* transmits a bit, hence bit by bit                                       */
/*=====================================================*/
int receive(int r_bit, int br, int nr, int npid, int ndata, int pdflag)
{
 unsigned char checksum;
 static unsigned check;
 switch (pos)
  {
  case 0: /* reception of the MID bits */
          if(br == 10) {
                receiver.frame.start_bit |= (r_bit);
                if(receiver.frame.start_bit != 0)
                  return(2);
          }
          else if(br == 9) {
                receiver.frame.stop_bit |= (r_bit);
                if(receiver.frame.stop_bit != 1)
                  return(2);
          }
          else receiver.frame.MID |= (r_bit << br);
          if(br == 0) check = receiver.frame.MID;
        return(SUCCESS);
        break;

case 1:  /* reception of the PID and data bits */
        if(pdflag == 0){
                if(br == 10) {
                        receiver.frame.start_bit |= (r_bit);
                        if(receiver.frame.start_bit != 0)
                                return(2);
                }
                else if(br == 9) {
                        receiver.frame.stop_bit |= (r_bit);
                        if(receiver.frame.stop_bit != 1)
                                return(2);
                }
                else receiver.frame.PID[npid] |= (r_bit << br);
                if(br == 0) check += receiver.frame.PID[npid];
        }
        else {
                if(br == 10) {
                        receiver.frame.start_bit |= (r_bit);
                        if(receiver.frame.start_bit != 0)
                                return(2);
                }
                else if(br == 9) {
                        receiver.frame.stop_bit |= (r_bit);
                        if(receiver.frame.stop_bit != 1)
                                return(2);
                }
                else receiver.frame.DATA[npid][ndata] |= (r_bit << br);
                if(br == 0) check += receiver.frame.DATA[npid][ndata];
        }
        return(SUCCESS);
        break;

case 2:  /* reception of the checksum bits */
        if(br == 10) {
                receiver.frame.start_bit |= (r_bit);
                if(receiver.frame.start_bit != 0)
                        return(2);
        }
        else if(br == 9) {
                receiver.frame.stop_bit |= (r_bit);
                if(receiver.frame.stop_bit != 1)
                        return(2);
                tic+=10;
```

```
               idle_time+=10;
          }
          else receiver.frame.CHECKSUM |= (r_bit << br);
          if(br == 0)
               checksum = ~check + 1;

          /* to check for error */
          if((checksum != receiver.frame.CHECKSUM) && (br==0))
               return(2);
          else 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)
{
 static long bit_tic = -1;

 if (g_val & (1 << bits))
   {
   if (rand_rate(errorRate))
     {
     bus_value = 0;
     bit_tic = tic;
     return(0);
     }
   if (bit_tic==tic) {
     bus_value &=1;
     return(1);
     }
   else
     {
     bus_value = 1;
     bit_tic = tic;
     return(1);
```

```
      }
    }
  else
    {
   if (rand_rate(errorRate))
     if (bit_tic==tic)
       {
       bus_value &=1;
       return(1);
       }
     else
       {
       bus_value = 1;
       bit_tic = tic;
       return(1);
       }
   bit_tic = tic;
   bus_value = 0;
   return(0);
    }
}

/*===================================================*/
/*  This function transmits a message over the bus.  The sender        */
/*  station transmits bit by bit. It starts of by transmitting the     */
/*  stop bit, followed by the character which is also transmitted bit  */
/*  by bit, and then finally ends with a stop bit. In this way it      */
/*  transmits the MID, PID, DATA and the CHECKSUM                      */
/*===================================================*/

int transmit()
{
 int  i,b,r, k, bit_val, bit_flag, pid_data;
 unsigned char val;

 /* bus is held by the current transmitter */
 bus_flag = BUSY;
 i=k=0;

 while (pos < 3)
  {
   switch (pos)
    {

    case 0:  /* transmission of MID bits */
```

```
        val = node[n].fram[k].MID;
              bit_val = get_bit(0,0);
         if ((bit_flag = msg_filter(bit_val,10,0,0,0,0)) != SUCCESS)
           return(bit_flag);
        for (b = 7;b >= 0;b--)
          {
          bit_val = get_bit(val, b);
          if ((bit_flag =
                       msg_filter(bit_val,b,0,0,0,0)) != SUCCESS)
            return(bit_flag);
          }
              bit_val = get_bit(1,0);
         if ((bit_flag =
                       msg_filter(bit_val,9,0,0,0,0)) != SUCCESS)
           return(bit_flag);
       pos++;
       break;

case 1:  /* transmission of PID and data bits */
              for(i=0;i<node[n].fram[k].total_no_of_parameters;i++) {
               bit_val = get_bit(0,0);
         if ((bit_flag =
                       msg_filter(bit_val,10,0,0,0,0)) != SUCCESS)
           return(bit_flag);
        val = node[n].fram[k].PID[i];
        for (b = 7;b >= 0;b--)
          {
          bit_val = get_bit(val, b);
                pid_data = 0;
          if ((bit_flag =
                       msg_filter(bit_val,b,0,i,0,pid_data)) != SUCCESS)
            return(bit_flag);
          }
               bit_val = get_bit(1,0);
         if ((bit_flag =
                       msg_filter(bit_val,9,0,0,0,0)) != SUCCESS)
           return(bit_flag);
                for(r=0;r<node[n].fram[k].tno_of_data[i];r++) {
                       bit_val = get_bit(0,0);
             if((bit_flag=
                          msg_filter(bit_val,10,0,0,0,0))!= SUCCESS)
                       return(bit_flag);
           val = node[n].fram[k].DATA[i][r];
           for (b = 7;b >= 0;b--)
           {
              bit_val = get_bit(val, b);
                    pid_data = 1;
              if((bit_flag=
                          msg_filter(bit_val,b,0,i,r,pid_data))!=SUCCESS)
                return(bit_flag);
             }
                      bit_val = get_bit(1,0);
               if((bit_flag=
                           msg_filter(bit_val,9,0,0,0,0))!= SUCCESS)
                      return(bit_flag);
                  }
                }
         pos++;
         break;

    case 2:  /* transmission of CHECKSUM bits */
            bit_val = get_bit(0,0);
         if((bit_flag =
                       msg_filter(bit_val,10,0,0,0,0))!= SUCCESS)
        return(bit_flag);
         val = node[n].fram[k].CHECKSUM;
         for (b = 7;b >= 0;b--)
           {
          bit_val = get_bit(val, b);
          if ((bit_flag =
                       msg_filter(bit_val,b,0,0,0,0)) != SUCCESS)
            return(bit_flag);
           }
            bit_val = get_bit(1,0);
         if((bit_flag =
                       msg_filter(bit_val,9,0,0,0,0))!= SUCCESS)
        return(bit_flag);
         pos++;
         break;
   }
  }
 /* end of message transmission */
 return(OVER);
}

/*==================================================*/
/* This function performs message filtering within the J1587 nodes.       */
/*==================================================*/

int msg_filter(int bit_val, int bm, int nm,int pm, int dm, int pd)
```

```
{
 int bit_read;

 bit_read = receive(bit_val,bm,nm,pm,dm,pd);

 if (bit_read != SUCCESS)
   {
   return(bit_read);
   }
 else
   {
   msg_time++;
   inc_tic(1);
   }
 return(SUCCESS);
}

/*==========================================================*/
/*  This module performs the initiation of a message transfer.            */
/*  It checks for a successful message transfer and form errors           */
/*==========================================================*/

void msg_transfer(int mode)
{
 int i;

 while (mode < 3)
   {
   switch (mode)
     {
     case 0:  /* action after a successful message transfer */
            if (runtimeOutput)
              {
              fprintf(fp_out,
                     "\n*** Message %d of Node %d successfully TX ",m,n);
              fprintf(fp_out,"at time %ld\n",tic);
              }
            response_time += (tic - msg_time) - node[n].msg[m].release;
            node[n].response_time +=(tic - msg_time)-node[n].msg[m].release;
            turnaround_time += tic - node[n].msg[m].release;
            node[n].turnaround_time += tic - node[n].msg[m].release;
            if (tic <= node[n].msg[m].deadline)
              {
              /* message was delivered on time (with slack to spare) */
              }
```

```
          else
           {
           /* message was not delivered on time */
           missed_msgs++;
           node[n].missed_msgs++;
           }
          data_time += msg_time;
          tx_msgs++;
          node[n].tx_msgs++;

          /* Add node stats */

          /* Prepare for next tx */
          msg_time = 0;
          node[n].transfer = NO;
          node[n].msg[m].error_flag = NO;
            node[n].msg[m].release += node[n].msg[m].period;
          node[n].msg[m].deadline += node[n].msg[m].period;
          receiver_frame_init();

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

     case 2:  /* action after a form error occurs */
          receiver_frame_init();
          retx_msgs++;
          form_count++;
          if (runtimeOutput) {
           fprintf(fp_out,
                   "\n\tFORM ERROR in Message %d of Node %d ",m, n);
           fprintf(fp_out,"at time %ld\n\n",tic);
          }
            error_msgs++;
          node[n].error_msgs++;
          node[n].msg[m].error_flag = YES;
            msg_time = 0;
```

```
                return;
        default: stopsim();
                return;

    }
  }
}

/*=======================================================*/
/* This function is used to read a particular priority for the first         */
/* PID of the message.                                                       */
/*=======================================================*/

int pread_file(int sent_pid)
{
   char pid[8],pri[8], S[80];
   int len,priority,check_pid;

   len=check_pid=priority=0;

   while(!feof(pin) && (check_pid != sent_pid)) {
          fgets(S,79,pin);
          len=strlen(S);
          S[len-1] = '\0';
          sscanf(S,"%s %s",pid,pri);
          check_pid = atoi(pid);
          priority = atoi(pri);
   }
   return(priority);
}

/*=======================================================*/
/* This initializes the receiver parameters before receiving thc            */
/* transmitted bits                                                          */
/*=======================================================*/
void receiver_frame_init()
{
 int i,j;

 receiver.frame.start_bit = 0;
 receiver.frame.MID = 0;
 for (i=0; i<MAX_NO_OF_PID; i++) {
         receiver.frame.PID[i] = 0;
         for(j=0;j<MAX_NO_OF_DATA;j++)
         receiver.frame.DATA[i][j] = 0;
 }
 receiver.frame.CHECKSUM = 0;
 receiver.frame.stop_bit = 1;
 return;
}
/*==================================================*/
/*=== END FILE: sim.c====================================*/
```

APPENDIX D

INPUT FILES

NUMBER OF NODES = 2 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES2.dat==========================================*/
1
1234
10000
9600
500
10000
0.0001
2
1
1
50
9900
0
400
100
2
85 150
20 220
2
1
100
9999
0
600
200
2
20 210
21 212
/*==END FILE: NODES2.dat======================================*/
```

NUMBER OF NODES = 5 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODE5.dat==========================================*/
1
1234
10000
9600
500
10000
0.0001
5
1
1
10
5200
0
900
100
2
85 150
20 220
2
1
50
5400
0
800
200
2
20 210
21 212
3
1
250
5900
0
990
300
2
30 310
31 312
4
1
10
5500
```

```
0
1900
400
2
40 410
41 411
5
1
10
5700
0
870
500
1
50 510
/*==END FILE: NODES5.dat======================================*/
```

NUMBER OF NODES = 10 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES10.dat========================================*/
1
1234
10000
9600
500
10000
0.0001
10
1
1
1100
9900
0
1900
100
2
11 111
12 112
2
2
900
999
0
1900
210
2
21 211
22 212
300
490
0
1980
199
3
42 20
52 220
62 220
3
1
310
```

```
500
0
980
300
1
31 311
4
2
100
250
0
1900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
5
1
100
9700
0
1500
500
1
51 511
6
1
1000
9900
0
1400
600
1
61 611
7
1
```

```
400
9900
0
1530
700
1
71 711
8
1
1000
9900
0
850
800
3
81 811
82 812
83 813
9
1
300
9800
0
1500
900
2
91 911
92 912
10
1
900
9895
0
1500
1000
1
101 1011
/*==END FILE: NODES10.dat======================================*/
```

NUMBER OF NODES = 20 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES20.dat========================================*/
1
1234
10000
9600
500
10000
0.0001
20
1
1
100
9900
0
7999
100
2
 11 111
 12 112
 2
 1
 1000
 9099
 0
 1990
 210
 2
 21 211
 22 212
 3
 1
 310
 500
 0
 8980
 300
 1
 31 311
 4
 2
 1100
 9250
 0
```

```
7990
400
2
41 411
42 412
300
380
0
8990
401
3
43 413
44 414
45 415
5
1
1100
9700
0
7999
500
1
51 511
6
1
1000
9900
0
1999
600
1
61 611
7
1
400
9900
0
1999
700
1
71 711
8
1
1000
9900
```

```
0
1850
800
3
81 811
82 812
83 813
9
1
900
9800
0
1900
900
2
91 911
92 912
10
1
900
9895
0
1999
1000
1
101 1011
11
1
1500
9999
0
1900
100
2
11 111
12 112
12
2
1100
9200
0
9900
210
2
21 211
```

```
22 212
400
490
0
1980
199
3
42 20
52 220
62 220
13
1
1310
9500
0
4980
300
1
31 311
14
2
150
250
0
9999
400
2
41 411
42 412
180
380
0
1999
401
3
43 413
44 414
45 415
15
1
1180
9700
0
1500
500
```

```
1
51 511
16
1
1000
9900
0
1999
600
1
61 611
17
1
1400
9900
0
1530
700
1
71 711
18
1
1200
9900
0
7850
800
3
81 811
82 812
83 813
19
1
300
9800
0
1999
900
2
91 911
92 912
20
1
900
1895
```

```
0
1500
1000
1
101 1011
/*==END FILE: NODES20.dat====================================*/
```

NUMBER OF NODES = 30 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES30.dat========================================*/
1
1234
10000
9600
500
10000
0.0001
30
1
1
100
9900
0
7999
100
2
11 111
12 112
2
1
1000
9099
0
1990
210
2
21 211
22 212
3
1
310
500
0
8980
300
1
31 311
4
2
1100
9250
0
```

```
7990
400
2
41 411
42 412
300
380
0
8990
401
3
43 413
44 414
45 415
5
1
1100
9700
0
7999
500
1
51 511
6
1
1000
9900
0
1999
600
1
61 611
7
1
400
9900
0
1999
700
1
71 711
8
1
130
9900
```

```
0
1850
800
3
81 811
82 812
83 813
9
1
900
9800
0
1900
900
2
91 911
92 912
10
1
900
9895
0
9500
1000
1
101 1011
11
1
500
9999
0
1900
100
2
11 111
12 112
12
2
1100
9200
0
9900
210
2
21 211
```

```
22 212
400
490
0
1980
199
3
42 20
52 220
62 220
13
1
310
9500
0
4980
300
1
31 311
14
2
150
250
0
9999
400
2
41 411
42 412
180
380
0
9999
401
3
43 413
44 414
45 415
15
1
180
9700
0
1500
500
```

```
1
51 511
16
1
120
9900
0
1999
600
1
61 611
17
1
400
9900
0
8530
700
1
71 711
18
1
200
9900
0
7850
800
3
81 811
82 812
83 813
19
1
300
9800
0
1999
900
2
91 911
92 912
20
1
150
1895
```

```
0
1500
1000
1
101 1011
21
1
200
9900
0
1900
100
2
11 111
12 112
22
2
900
999
0
9900
210
2
21 211
22 212
300
490
0
8980
199
3
42 20
52 220
62 220
23
1
310
500
0
9980
300
1
31 311
24
2
```

```
100
250
0
9900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
25
1
100
9700
0
9500
500
1
51 511
26
1
110
9900
0
8400
600
1
61 611
27
1
400
9900
0
1530
700
1
71 711
28
```

```
1
100
9900
0
9850
800
3
81 811
82 812
83 813
29
1
300
9800
0
8500
900
2
91 911
92 912
30
1
300
9895
0
1500
1000
1
101 1011
/*==END FILE: NODES30.dat====================================*/
```

NUMBER OF NODES = 40 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES40.dat=========================================*/
1
1234
10000
9600
500
10000
0.0001
40
1
1
100
9900
0
7999
100
2
11 111
12 112
2
1
1000
9099
0
1990
210
2
21 211
22 212
3
1
310
500
0
8980
300
1
31 311
4
2
1100
9250
0
```

```
7990
400
2
41 411
42 412
300
380
0
8990
401
3
43 413
44 414
45 415
5
1
1100
9700
0
7999
500
1
51 511
6
1
1000
9900
0
1999
600
1
61 611
7
1
400
9900
0
1999
700
1
71 711
8
1
1000
9900
```

```
0
8850
800
3
81 811
82 812
83 813
9
1
900
9800
0
1900
900
2
91 911
92 912
10
1
900
9895
0
9500
1000
1
101 1011
11
1
1500
9999
0
1900
100
2
11 111
12 112
12
2
1100
9200
0
9900
210
2
21 211
```

```
22 212
400
490
0
1980
199
3
42 20
52 220
62 220
13
1
1310
9500
0
4980
300
1
31 311
14
2
150
250
0
9999
400
2
41 411
42 412
180
380
0
9999
401
3
43 413
44 414
45 415
15
1
1180
9700
0
1500
500
```

```
1
51 511
16
1
1000
9900
0
1999
600
1
61 611
17
1
1400
9900
0
8530
700
1
71 711
18
1
1200
9900
0
7850
800
3
81 811
82 812
83 813
19
1
300
9800
0
1999
900
2
91 911
92 912
20
1
900
1895
```

```
0
1500
1000
1
101 1011
21
1
1200
9900
0
8900
100
2
11 111
12 112
22
2
900
999
0
9900
210
2
21 211
22 212
300
490
0
8980
199
3
42 20
52 220
62 220
23
1
310
500
0
9980
300
1
31 311
24
2
```

```
100
250
0
9900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
25
1
100
9700
0
9500
500
1
51 511
26
1
1000
9900
0
8400
600
1
61 611
27
1
400
9900
0
9530
700
1
71 711
28
```

```
1
1000
9900
0
9850
800
3
81 811
82 812
83 813
29
1
300
9800
0
8500
900
2
91 911
92 912
30
1
900
9895
0
1500
1000
1
101 1011
31
1
1100
9900
0
1900
100
2
11 111
12 112
32
2
900
999
0
8900
```

```
210
2
21 211
22 212
300
490
0
1980
199
3
42 20
52 220
62 220
33
1
310
500
0
8980
300
1
31 311
34
2
100
250
0
8900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
35
1
100
9700
```

```
0
1500
500
1
51 511
36
1
1000
9900
0
1400
600
1
61 611
37
1
400
9900
0
8530
700
1
71 711
38
1
1000
9900
0
8850
800
3
81 811
82 812
83 813
39
1
300
9800
0
1500
900
2
91 911
92 912
40
```

```
1
900
9895
0
1500
1000
1
101 1011
/*==END FILE: NODES40.dat===================================*/
```

NUMBER OF NODES = 50 AND SIM_TIME = 10000 BIT TIMES

```
/*==FILE: NODES50.dat========================================*/
1
1234
10000
9600
500
10000
0.0001
50
1
1
100
9900
0
7999
100
2
11 111
12 112
2
1
1000
9099
0
1990
210
2
21 211
22 212
3
1
310
500
0
8980
300
1
31 311
4
2
1100
9250
0
```

```
7990
400
2
41 411
42 412
300
380
0
8990
401
3
43 413
44 414
45 415
5
1
1100
9700
0
7999
500
1
51 511
6
1
1000
9900
0
1999
600
1
61 611
7
1
400
9900
0
1999
700
1
71 711
8
1
1000
9900
```

```
0
8850
800
3
81 811
82 812
83 813
9
1
900
9800
0
1900
900
2
91 911
92 912
10
1
900
9895
0
9500
1000
1
101 1011
11
1
1500
9999
0
1900
100
2
11 111
12 112
12
2
1100
9200
0
9900
210
2
21 211
```

```
22 212
400
490
0
1980
199
3
42 20
52 220
62 220
13
1
1310
9500
0
4980
300
1
31 311
14
2
150
250
0
9999
400
2
41 411
42 412
180
380
0
9999
401
3
43 413
44 414
45 415
15
1
1180
9700
0
1500
500
```

```
1
51 511
16
1
1000
9900
0
1999
600
1
61 611
17
1
1400
9900
0
8530
700
1
71 711
18
1
1200
9900
0
7850
800
3
81 811
82 812
83 813
19
1
300
9800
0
1999
900
2
91 911
92 912
20
1
900
1895
```

```
0
1500
1000
1
101 1011
21
1
1200
9900
0
8900
100
2
11 111
12 112
22
2
900
999
0
9900
210
2
21 211
22 212
300
490
0
8980
199
3
42 20
52 220
62 220
23
1
310
500
0
9980
300
1
31 311
24
2
```

```
100
250
0
9900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
25
1
100
9700
0
9500
500
1
51 511
26
1
1000
9900
0
8400
600
1
61 611
27
1
400
9900
0
9530
700
1
71 711
28
```

```
1
1000
9900
0
9850
800
3
81 811
82 812
83 813
29
1
300
9800
0
8500
900
2
91 911
92 912
30
1
900
9895
0
1500
1000
1
101 1011
31
1
1100
9900
0
1900
100
2
11 111
12 112
32
2
900
999
0
8900
```

```
210
2
21 211
22 212
300
490
0
1980
199
3
42 20
52 220
62 220
33
1
310
500
0
8980
300
1
31 311
34
2
100
250
0
8900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
35
1
100
9700
```

```
0
1500
500
1
51 511
36
1
1000
9900
0
1400
600
1
61 611
37
1
400
9900
0
8530
700
1
71 711
38
1
1000
9900
0
8850
800
3
81 811
82 812
83 813
39
1
300
9800
0
1500
900
2
91 911
92 912
40
```

```
1
900
9895
0
1500
1000
1
101 1011
41
1
1100
9900
0
1900
100
2
11 111
12 112
42
2
900
999
0
1900
210
2
21 211
22 212
300
490
0
1980
199
3
42 20
52 220
62 220
43
1
310
500
0
980
300
1
```

```
31 311
44
2
100
250
0
1900
400
2
41 411
42 412
100
380
0
1960
401
3
43 413
44 414
45 415
45
1
100
9700
0
1500
500
1
51 511
46
1
1000
9900
0
7400
600
1
61 611
47
1
400
9900
0
1530
700
```

```
1
71 711
48
1
1000
9900
0
850
800
3
81 811
82 812
83 813
49
1
300
9800
0
1500
900
2
91 911
92 912
50
1
900
9895
0
1500
1000
1
101 1011
/*==END FILE: NODES50.dat===================================*/
```

APPENDIX E

STATISTICS FILE

NUMBER OF NODES = 50 and SIM_TIME = 10000 bit times

```
------------------------------------------------------------------------------------------
           Network Statistics
------------------------------------------------------------------------------------------

Number of nodes = 50
Number of messages transmitted = 159

Total number of error messages        =       2
Total number of collisions            =     113
Idle time in the network              =  0.16563 sec
Busy time in the network              =  0.83240 sec
Average response time                 =  0.27335 sec
Simulation time                       =  0.98958 sec
Network load                          =  84.1158 %
Network throughput                    = 8407.5791 bits/sec
------------------------------------------------------------------------------------------
           Node Statistics
------------------------------------------------------------------------------------------
Node       No of   No of   No of   No of          Average
           msgs    error   coll.   deadlines      response
           sent    msgs            missed         time
------------------------------------------------------------------------------------------
1          1       0       0       0              0.01521
2          5       0       0       0              0.44933
3          1       0       0       1              0.09521
4          2       0       0       1              0.10292
5          1       0       3       0              0.15875
6          3       0       2       0              0.26330
7          3       0       2       0              0.25865
8          1       0       0       0              0.17854
9          4       0       0       0              0.27323
10         1       0       2       0              0.17000
11         3       0       0       0              0.29701
12         4       0       4       3              0.25297
13         1       0       0       0              0.24104
14         2       0       0       2              0.05458
15         4       0       0       0              0.29547
16         3       0       2       0              0.28649
17         1       0       0       0              0.24625
```

18	1	0	0	0	0.29937
19	4	0	1	0	0.25766
20	4	0	3	4	0.26385
21	1	0	1	0	0.30875
22	2	0	6	2	0.13969
23	1	0	2	1	0.11187
24	5	0	2	3	0.17858
25	1	0	2	0	0.03125
26	1	0	0	0	0.20917
27	1	0	0	0	0.20146
28	1	0	2	0	0.21250
29	1	0	3	0	0.08812
30	4	0	2	0	0.27870
31	4	1	2	0	0.31500
32	5	0	9	5	0.26192
33	1	0	0	1	0.15979
34	5	0	7	4	0.18487
35	5	0	5	0	0.24896
36	5	0	0	0	0.29558
37	1	0	1	0	0.20562
38	1	0	2	0	0.25604
39	6	0	5	0	0.42378
40	4	0	0	0	0.28901
41	3	0	0	0	0.28146
42	7	0	7	7	0.28000
43	6	0	4	6	0.28917
44	8	0	10	8	0.23271
45	5	0	5	0	0.25438
46	1	0	0	0	0.27646
47	7	0	4	0	0.41024
48	7	0	2	0	0.29247
49	4	1	6	0	0.25344
50	7	0	5	0	0.41976

APPENDIX F

OUTPUT FILE

NUMBER OF NODES = 50 AND SIM_TIME = 10000 Bit Times

```
11110100100001000101 tic=19
00110100110111000101 tic=39
01010100111001011100 tic=59
1101
*** Message 0 of Node 24 successfully TX at time 74
010010 tic=79
00010001010011010011 tic=99
01110001010101010011 tic=119
10010111001101
*** Message 1 of Node 34 successfully TX at time 144
0011001001000001 tic=159
01110011011111000001 tic=179
10010011100001010100 tic=199
1101
*** Message 0 of Node 1 successfully TX at time 214
010010 tic=219
00010001010011010011 tic=239
01110001010101010011 tic=259
10010111001101
*** Message 1 of Node 24 successfully TX at time 284
0100100001000101 tic=299
00110100110111000101 tic=319
01010100111001011100 tic=339
1101
*** Message 0 of Node 34 successfully TX at time 354
011110 tic=359
10010001100111011111 tic=379
11110110110101
*** Message 0 of Node 25 successfully TX at time 404
0111101001000110 tic=419
01110111111111011011 tic=439
0101
*** Message 0 of Node 35 successfully TX at time 454
010010 tic=459
00010001010011010011 tic=479
01110001010101010011 tic=499
10010111001101
*** Message 0 of Node 44 successfully TX at time 524
0111101001000110 tic=539
```

```
01110111111111011011 tic=559
0101
*** Message 0 of Node 45 successfully TX at time 574
010010 tic=579
00010001010011010011 tic=599
01110001010101010011 tic=619
10010111001101
*** Message 1 of Node 44 successfully TX at time 644
0100100001000101 tic=659
00110100110111000101 tic=679
01010100111001011100 tic=699
1101
*** Message 0 of Node 14 successfully TX at time 714
010010 tic=719
00010001010011010011 tic=739
01110001010101010011 tic=759
10010111001101
*** Message 1 of Node 14 successfully TX at time 784
0100100001000101 tic=799
00110100110111000101 tic=819
01010100111001011100 tic=839
1101
*** Message 1 of Node 4 successfully TX at time 854
010000 tic=859
10010010110111010001 tic=879
11110010111001010010 tic=899
00010101001101
*** Message 0 of Node 19 successfully TX at time 924
0110100101000010 tic=939
10110110100111000010 tic=959
11010110101001001011 tic=979
1001
*** Message 1 of Node 22 successfully TX at time 994
010000 tic=999
10010010110111010001 tic=1019
11110010111001010010 tic=1039
00010101001101
*** Message 0 of Node 29 successfully TX at time 1064
0001011001000011 tic=1079
11110001101111001111 tic=1099
1101
*** Message 0 of Node 3 successfully TX at time 1114
011010 tic=1119
01010000101011011010 tic=1139
01110000101101011010 tic=1159
```

```
10010010111001
*** Message 1 of Node 42 successfully TX at time 1184
0100001001001011 tic=1199
01110100011111001011 tic=1219
10010100100001010100 tic=1239
1101
*** Message 0 of Node 39 successfully TX at time 1254
000101 tic=1259
10010000111111000110 tic=1279
11110011111101
*** Message 0 of Node 23 successfully TX at time 1304
0110100101000010 tic=1319
10110110100111000010 tic=1339
11010110101001001011 tic=1359
1001
*** Message 1 of Node 32 successfully TX at time 1374
011010 tic=1379
01010000101011011010 tic=1399
01110000101101011010 tic=1419
10010010111001
*** Message 0 of Node 2 successfully TX at time 1444
0101111001001000 tic=1459
11110110001111000110 tic=1479
1101
*** Message 0 of Node 7 successfully TX at time 1494
001011 tic=1499
00010001111011001100 tic=1519
01110000010001
*** Message 0 of Node 6 successfully TX at time 1544
0100100001000101 tic=1559
00110100110111000101 tic=1579
01010100111001011100 tic=1599
1101
*** Message 0 of Node 4 successfully TX at time 1614
010000 tic=1619
10010010110111010001 tic=1639
11110010111001010010 tic=1659
00010101001101
*** Message 0 of Node 49 successfully TX at time 1684
0110100101000010 tic=1699
10110110100111000010 tic=1719
11010110101001001011 tic=1739
1001
*** Message 1 of Node 12 successfully TX at time 1754
011110 tic=1759
```

```
10010001100111011111 tic=1779
11110110110101
*** Message 0 of Node 5 successfully TX at time 1804
0001011001000011 tic=1819
11110001101111001111 tic=1839
1101
*** Message 0 of Node 33 successfully TX at time 1854
010000 tic=1859
10010010110111010001 tic=1879
11110010111001010010 tic=1899
00010101001101
*** Message 0 of Node 9 successfully TX at time 1924
0111010001001100 tic=1939
10110111100111011000 tic=1959
0001
*** Message 0 of Node 10 successfully TX at time 1974
011101 tic=1979
00010011001011011110 tic=1999
01110110000001
*** Message 0 of Node 20 successfully TX at time 2024
0001000001001010 tic=2039
00110001010111001010 tic=2059
01010001011001001010 tic=2079
01110001011011001100 tic=2099
1101
*** Message 0 of Node 8 successfully TX at time 2114
000101 tic=2119
10010000111111000110 tic=2139
11110011111101
*** Message 0 of Node 43 successfully TX at time 2164
0010110001000111 tic=2179
10110011000111000001 tic=2199
0001
*** Message 0 of Node 16 successfully TX at time 2214
011010 tic=2219
01010000101011011010 tic=2239
01110000101101011010 tic=2259
10010010111001
*** Message 0 of Node 22 successfully TX at time 2284
0101111001001000 tic=2299
11110110001111000110 tic=2319
1101
*** Message 0 of Node 27 successfully TX at time 2334
010111 tic=2339
10010010001111011000 tic=2359
```

```
11110001101101
*** Message 0 of Node 37 successfully TX at time 2384
0010110001000111 tic=2399
10110011000111000001 tic=2419
0001
*** Message 0 of Node 26 successfully TX at time 2434
000100 tic=2439
00010010100011000101 tic=2459
01110010100101000101 tic=2479
10010010100111000101 tic=2499
10110011001101
*** Message 0 of Node 28 successfully TX at time 2524
0111010001001100 tic=2539
10110111100111011000 tic=2559
0001
*** Message 0 of Node 30 successfully TX at time 2574
010111 tic=2579
10010010001111011000 tic=2599
11110001101101
*** Message 0 of Node 47 successfully TX at time 2624
0110100101000010 tic=2639
10110110100111000010 tic=2659
11010110101001001011 tic=2679
1001
*** Message 0 of Node 12 successfully TX at time 2694
001100 tic=2699
10010000010111001101 tic=2719
11110000011001001110 tic=2739
00010101001101
*** Message 0 of Node 31 successfully TX at time 2764
0110100101000010 tic=2779
10110110100111000010 tic=2799
11010110101001001011 tic=2819
1001
*** Message 0 of Node 32 successfully TX at time 2834
011101 tic=2839
00010011001011011110 tic=2859
01110110000001
*** Message 0 of Node 40 successfully TX at time 2884
0010110001000111 tic=2899
10110011000111000001 tic=2919
0001
*** Message 0 of Node 36 successfully TX at time 2934
000100 tic=2939
00010010100011000101 tic=2959
```

```
01110010100101000101 tic=2979
10010010100111000101 tic=2999
10110011001101
*** Message 0 of Node 38 successfully TX at time 3024
0110100101000010 tic=3039
10110110100111000010 tic=3059
11010110101001001011 tic=3079
1001
*** Message 0 of Node 42 successfully TX at time 3094
011101 tic=3099
00010011001011011110 tic=3119
01110110000001
*** Message 0 of Node 50 successfully TX at time 3144
0010110001000111 tic=3159
10110011000111000001 tic=3179
0001
*** Message 0 of Node 46 successfully TX at time 3194
000100 tic=3199
00010010100011000101 tic=3219
01110010100101000101 tic=3239
10010010100111000101 tic=3259
10110011001101
*** Message 0 of Node 48 successfully TX at time 3284
0011001001000001 tic=3299
01110011011111000001 tic=3319
10010011100001010100 tic=3339
1101
*** Message 0 of Node 41 successfully TX at time 3354
011110 tic=3359
10010001100111011111 tic=3379
11110110110101
*** Message 0 of Node 15 successfully TX at time 3404
0001000001001010 tic=3419
00110001010111001010 tic=3439
01010001011001001010 tic=3459
01110001011011001100 tic=3479
1101
*** Message 0 of Node 18 successfully TX at time 3494
001100 tic=3499
10010000010111001101 tic=3519
11110000011001001110 tic=3539
00010101001101
*** Message 0 of Node 21 successfully TX at time 3564
0001011001000011 tic=3579
11110001101111001111 tic=3599
```

```
1101
*** Message 0 of Node 43 successfully TX at time 3614
000101 tic=3619
10010000111111000110 tic=3639
11110011111101
*** Message 0 of Node 13 successfully TX at time 3664
0001000001001010 tic=3679
00110001010111001010 tic=3699
01010001011001001010 tic=3719
01110001011011001100 tic=3739
1101
*** Message 0 of Node 48 successfully TX at time 3754
010111 tic=3759
10010010001111011000 tic=3779
11110001101101
*** Message 0 of Node 17 successfully TX at time 3804
0011001001000001 tic=3819
01110011011111000001 tic=3839
10010011100001010100 tic=3859
1101
*** Message 0 of Node 11 successfully TX at time 3874
011110 tic=3879
10010001100111011111 tic=3899
11110110110101
*** Message 0 of Node 35 successfully TX at time 3924
0111101001000110 tic=3939
01110111111111011011 tic=3959
0101
*** Message 0 of Node 45 successfully TX at time 3974
010000 tic=3979
10010010110111010001 tic=3999
11110010111001010010 tic=4019
00010101001101
*** Message 0 of Node 39 successfully TX at time 4044
0100001001001011 tic=4059
01110100011111001011 tic=4079
10010100100001010100 tic=4099
1101
*** Message 0 of Node 49 successfully TX at time 4114
011101 tic=4119
00010011001011011110 tic=4139
01110110000001
*** Message 0 of Node 20 successfully TX at time 4164
0010110001000111 tic=4179
10110011000111000001 tic=4199
```

```
0001
*** Message 0 of Node 36 successfully TX at time 4214
011101 tic=4219
00010011001011011110 tic=4239
01110110000001
*** Message 0 of Node 30 successfully TX at time 4264
0101111001001000 tic=4279
11110110001111000110 tic=4299
1101
*** Message 0 of Node 47 successfully TX at time 4314
010010 tic=4319
00010001010011010011 tic=4339
01110001010101010011 tic=4359
10010111001101
*** Message 0 of Node 44 successfully TX at time 4384
0111010001001100 tic=4399
10110111100111011000 tic=4419
0001
*** Message 0 of Node 40 successfully TX at time 4434
010010 tic=4439
00010001010011010011 tic=4459
01110001010101010011 tic=4479
10010111001101
*** Message 1 of Node 24 successfully TX at time 4504
0100100001000101 tic=4519
00110100110111000101 tic=4539
01010100111001011100 tic=4559
1101
*** Message 1 of Node 34 successfully TX at time 4574
011101 tic=4579
00010011001011011110 tic=4599
01110110000001
*** Message 0 of Node 50 successfully TX at time 4624
0100100001000101 tic=4639
00110100110111000101 tic=4659
01010100111001011100 tic=4679
1101
*** Message 1 of Node 44 successfully TX at time 4694
011010 tic=4699
01010000101011011010 tic=4719
01110000101101011010 tic=4739
10010010111001
*** Message 1 of Node 42 successfully TX at time 4764
0100001001001011 tic=4779
01110100011111001011 tic=4799
```

```
10010100100001010100 tic=4819
1101
*** Message 0 of Node 19 successfully TX at time 4834
011010 tic=4839
01010000101011011010 tic=4859
01110000101101011010 tic=4879
10010010111001
*** Message 1 of Node 32 successfully TX at time 4904
0110100101000010 tic=4919
10110110100111000010 tic=4939
11010110101001001011 tic=4959
1001
*** Message 0 of Node 2 successfully TX at time 4974
000100 tic=4979
00010010100011000101 tic=4999
01110010100101000101 tic=5019
10010010100111000101 tic=5039
10110011001101
*** Message 0 of Node 48 successfully TX at time 5064
0100001001001011 tic=5079
01110100011111001011 tic=5099
10010100100001010100 tic=5119
1101
*** Message 0 of Node 9 successfully TX at time 5134
011010 tic=5139
01010000101011011010 tic=5159
01110000101101011010 tic=5179
10010010111001
*** Message 1 of Node 12 successfully TX at time 5204
0101111001001000 tic=5219
11110110001111000110 tic=5239
1101
*** Message 0 of Node 7 successfully TX at time 5254
001011 tic=5259
00010001111011001100 tic=5279
01110000010001
*** Message 0 of Node 6 successfully TX at time 5304
0001011001000011 tic=5319
11110001101111001111 tic=5339
1101
*** Message 0 of Node 43 successfully TX at time 5354
001011 tic=5359
00010001111011001100 tic=5379
01110000010001
*** Message 0 of Node 16 successfully TX at time 5404
```

```
0011011001000001 tic=5419
01110011011111000001 tic=5439
10010011100001010100 tic=5459
11
        FORM ERROR in Message 0 of Node 31 at time 5462

0011001001000000101 tic=5479
11001101111100000110 tic=5499
01001110000101010011 tic=5519
01
*** Message 0 of Node 31 successfully TX at time 5532
01101001 tic=5539
01000010101101101001 tic=5559
11000010110101101010 tic=5579
010010111001
*** Message 0 of Node 42 successfully TX at time 5602
0111101001000110001 tic=5619
11011111111101101101 tic=5639
01
*** Message 0 of Node 15 successfully TX at time 5652
00110010 tic=5659
01000001011100110111 tic=5679
11000001100100111000 tic=5699
010101001101
*** Message 0 of Node 41 successfully TX at time 5722
0111101001000110001 tic=5739
11011111111101101101 tic=5759
01
*** Message 0 of Node 35 successfully TX at time 5772
00010000 tic=5779
01001010001100010101 tic=5799
11001010010100010110 tic=5819
01001010011100010110 tic=5839
110011001101
*** Message 0 of Node 48 successfully TX at time 5862
0111101001000110001 tic=5879
11011111111101101101 tic=5899
01
*** Message 0 of Node 45 successfully TX at time 5912
01000010 tic=5919
01001011011101000111 tic=5939
11001011100101001000 tic=5959
010101001101
*** Message 0 of Node 39 successfully TX at time 5982
0100001001001011111 tic=5999
```

```
11010001111100101110 tic=6019
01010010000101010011 tic=6039

        FORM ERROR in Message 0 of Node 49 at time 6040

01000010010010110111 tic=6059
01000111110010111001 tic=6079
01001000010101001101
*** Message 0 of Node 49 successfully TX at time 6110
0001011001 tic=6119
00001111110001101111 tic=6139
0011111101 tic=6159

*** Message 0 of Node 43 successfully TX at time 6160
00101100010001111011 tic=6179
00110001110000010001
*** Message 0 of Node 36 successfully TX at time 6210
0111010001 tic=6219
00110010110111100111 tic=6239
0110000001 tic=6259

*** Message 0 of Node 20 successfully TX at time 6260
00110010010000010111 tic=6279
00110111110000011001 tic=6299
00111000010101001101
*** Message 0 of Node 11 successfully TX at time 6330
0111010001 tic=6339
00110010110111100111 tic=6359
0110000001 tic=6379

*** Message 0 of Node 30 successfully TX at time 6380
01110100010011001011 tic=6399
01111001110110000001
*** Message 0 of Node 40 successfully TX at time 6430
0101111001 tic=6439
00100011110110001111 tic=6459
0001101101 tic=6479

*** Message 0 of Node 47 successfully TX at time 6480
01110100010011001011 tic=6499
01111001110110000001
*** Message 0 of Node 50 successfully TX at time 6530
0100100001 tic=6539
00010100110100110111 tic=6559
00010101010100111001 tic=6579
```

```
0111001101 tic=6599

*** Message 0 of Node 44 successfully TX at time 6600
00010000010010100011 tic=6619
00010101110010100101 tic=6639
00010110010010100111 tic=6659
00010110110011001101
*** Message 0 of Node 48 successfully TX at time 6690
0100100001 tic=6699
00010100110100110111 tic=6719
00010101010100111001 tic=6739
0111001101 tic=6759

*** Message 1 of Node 24 successfully TX at time 6760
01001000010001010011 tic=6779
01001101110001010101 tic=6799
01001110010111001101
*** Message 1 of Node 34 successfully TX at time 6830
0100100001 tic=6839
00010100110100110111 tic=6859
00010101010100111001 tic=6879
0111001101 tic=6899

*** Message 1 of Node 44 successfully TX at time 6900
01000010010010110111 tic=6919
01000111110010111001 tic=6939
01001000010101001101
*** Message 0 of Node 9 successfully TX at time 6970
0110100101 tic=6979
00001010110110100111 tic=6999
00001011010110101001 tic=7019
0010111001 tic=7039

*** Message 1 of Node 42 successfully TX at time 7040
01101001010000101011 tic=7059
01101001110000101101 tic=7079
01101010010010111001
*** Message 1 of Node 32 successfully TX at time 7110
0100001001 tic=7119
00101101110100011111 tic=7139
00101110010100100001 tic=7159
0101001101 tic=7179

*** Message 0 of Node 19 successfully TX at time 7180
01101001010000101011 tic=7199
```

```
01101001110000101101 tic=7219
01101010010010111001
*** Message 0 of Node 2 successfully TX at time 7250
0111101001 tic=7259
00011001110111111111 tic=7279
0110110101 tic=7299

*** Message 0 of Node 15 successfully TX at time 7300
01101001010000101011 tic=7319
01101001110000101101 tic=7339
01101010010010111001
*** Message 1 of Node 12 successfully TX at time 7370
0011001001 tic=7379
00000101110011011111 tic=7399
00000110010011100001 tic=7419
0101001101 tic=7439

*** Message 0 of Node 31 successfully TX at time 7440
01011110010010001111 tic=7459
01100011110001101101
*** Message 0 of Node 7 successfully TX at time 7490
0001011001 tic=7499
00001111110001101111 tic=7519
0011111101 tic=7539

*** Message 0 of Node 43 successfully TX at time 7540
00101100010001111011 tic=7559
00110001110000010001
*** Message 0 of Node 6 successfully TX at time 7590
0110100101 tic=7599
00001010110110100111 tic=7619
00001011010110101001 tic=7639
0010111001 tic=7659

*** Message 0 of Node 42 successfully TX at time 7660
00101100010001111011 tic=7679
00110001110000010001
*** Message 0 of Node 16 successfully TX at time 7710
0111101001 tic=7719
00011001110111111111 tic=7739
0110110101 tic=7759

*** Message 0 of Node 35 successfully TX at time 7760
01111010010001100111 tic=7779
01111111110110110101
```

```
*** Message 0 of Node 45 successfully TX at time 7810
0100001001 tic=7819
00101101110100011111 tic=7839
00101110010100100001 tic=7859
0101001101 tic=7879

*** Message 0 of Node 39 successfully TX at time 7880
00101100010001111011 tic=7899
00110001110000010001
*** Message 0 of Node 36 successfully TX at time 7930
0100001001 tic=7939
00101101110100011111 tic=7959
00101110010100100001 tic=7979
0101001101 tic=7999

*** Message 0 of Node 49 successfully TX at time 8000
00010000010010100011 tic=8019
00010101110010100101 tic=8039
00010110010010100111 tic=8059
00010110110011001101
*** Message 0 of Node 48 successfully TX at time 8090
0111010001 tic=8099
00110010110111100111 tic=8119
0110000001 tic=8139

*** Message 0 of Node 20 successfully TX at time 8140
00110010010000010111 tic=8159
00110111110000011001 tic=8179
00111000010101001101
*** Message 0 of Node 41 successfully TX at time 8210
0111010001 tic=8219
00110010110111100111 tic=8239
0110000001 tic=8259

*** Message 0 of Node 30 successfully TX at time 8260
01110100010011001011 tic=8279
01111001110110000001
*** Message 0 of Node 40 successfully TX at time 8310
0101111001 tic=8319
00100011110110001111 tic=8339
0001101101 tic=8359

*** Message 0 of Node 47 successfully TX at time 8360
01110100010011001011 tic=8379
01111001110110000001
```

```
*** Message 0 of Node 50 successfully TX at time 8410
0111010001 tic=8419
00110010110111100111 tic=8439
0110000001 tic=8459

*** Message 0 of Node 50 successfully TX at time 8460
01011110010010001111 tic=8479
01100011110001101101
*** Message 0 of Node 47 successfully TX at time 8510
0111010001 tic=8519
00110010110111100111 tic=8539
0110000001 tic=8559

*** Message 0 of Node 50 successfully TX at time 8560
01011110010010001111 tic=8579
01100011110001101101
*** Message 0 of Node 47 successfully TX at time 8610
0001011001 tic=8619
00001111110001101111 tic=8639
0011111101 tic=8659

*** Message 0 of Node 43 successfully TX at time 8660
00110010010000010111 tic=8679
00110111110000011001 tic=8699
00111000010101001101
*** Message 0 of Node 11 successfully TX at time 8730
0001000001 tic=8739
00101000110001010111 tic=8759
00101001010001011001 tic=8779
00101001110001011011 tic=8799
0011001101 tic=8819

*** Message 0 of Node 48 successfully TX at time 8820
01111010010001100111 tic=8839
01111111110110110101
*** Message 0 of Node 15 successfully TX at time 8870
0100100001 tic=8879
00010100110100110111 tic=8899
00010101010100111001 tic=8919
0111001101 tic=8939

*** Message 0 of Node 44 successfully TX at time 8940
01110100010011001011 tic=8959
01111001110110000001
*** Message 0 of Node 50 successfully TX at time 8990
```

```
0100100001 tic=8999
00010100110100110111 tic=9019
00010101010100111001 tic=9039
0111001101 tic=9059

*** Message 1 of Node 24 successfully TX at time 9060
01001000010001010011 tic=9079
01001101110001010101 tic=9099
01001110010111001101
*** Message 1 of Node 34 successfully TX at time 9130
0101111001 tic=9139
00100011110110001111 tic=9159
0001101101 tic=9179

*** Message 0 of Node 47 successfully TX at time 9180
01001000010001010011 tic=9199
01001101110001010101 tic=9219
01001110010111001101
*** Message 1 of Node 44 successfully TX at time 9250
0100001001 tic=9259
00101101110100011111 tic=9279
00101110010100100001 tic=9299
0101001101 tic=9319

*** Message 0 of Node 9 successfully TX at time 9320
01111010010001100111 tic=9339
01111111110110110101
*** Message 0 of Node 35 successfully TX at time 9370
0111101001 tic=9379
00011001110111111111 tic=9399
0110110101 tic=9419

*** Message 0 of Node 45 successfully TX at time 9420
00101100010001111011 tic=9439
00110001110000010001
*** Message 0 of Node 36 successfully TX at time 9470
0110100101 tic=9479
00001010110110100111 tic=9499
00001011010110101001 tic=9519
0010111001 tic=9539

*** Message 1 of Node 42 successfully TX at time 9540
01101001010000101011 tic=9559
01101001110000101101 tic=9579
01101010010010111001
```

```
*** Message 1 of Node 32 successfully TX at time 9610
0011001001 tic=9619
00000101110011011111 tic=9639
00000110010011100001 tic=9659
0101001101 tic=9679

*** Message 0 of Node 31 successfully TX at time 9680
01000010010010110111 tic=9699
01000111110010111001 tic=9719
01001000010101001101
*** Message 0 of Node 19 successfully TX at time 9750
0110100101 tic=9759
00001010110110100111 tic=9779
00001011010110101001 tic=9799
0010111001 tic=9819

*** Message 0 of Node 2 successfully TX at time 9820
01000010010010110111 tic=9839
01000111110010111001 tic=9859
01001000010101001101
*** Message 0 of Node 39 successfully TX at time 9890
0100001001 tic=9899
00101101110100011111 tic=9919
00101110010100100001 tic=9939
0101001101 tic=9959

*** Message 0 of Node 39 successfully TX at time 9960
01101001010000101011 tic=9979
01101001110000101101 tic=9999
01101010010010111001
*** Message 0 of Node 2 successfully TX at time 10030

Stopped.
```

VITA 2

Viswanathan Anuradha

Candidate for the Degree of

Master of Science

Thesis: SIMULATION OF SAE J1587 STANDARD FOR ELECTRONIC DATA INTERCHANGE BETWEEN MICROCOMPUTER SYSTEMS IN HEAVY-DUTY VEHICLE APPLICATIONS

Major Field: Computer Science

Biographical:

Personal Data: Born in Bangalore, India on January 5, 1970, the daughter of R. Viswanathan and Shantha V.

Education: Received high school certificate from Baldwin Girls High School, Bangalore, India; Graduated from Dr. Ambedkar Institute of Technology, Bangalore, India in Aug. 1991; received Bachelor of Engineering Degree in Electronics and Communication Engineering. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1995.

Professional Experience: Graduate Research Assistant, Biosystems and Agricultural Engineering, Oklahoma State University, OK .