The Design and Implementation of CUENET: A Reconfigurable Network of Loosely Coupled Microcomputers

Clifford Grossner

A Thesis

in

The Department

of

Computer Science.

Presented in Partial Fulfillment of the Requirements

for the degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

September 1982

ABSTRACT

The Design and Implementation of CUENET: A Reconfigurable Network of Loosely Coupled Microcomputers

Clifford Grossner

There have been many efforts made by researchers to design multiple microcomputer systems in order to increase the data throughput of the total system. An essential requirement for parallel processing is an efficient interconnection mechanism. In this thesis we present the design and implementation of a multicomputer system based on a time shared bus called C-bus. This multicomputer system is referred to as CUENET (Concordia University Educational NETwork).

The computer interconnection topology on CUENET is reconfigurable under program control. A multicomputer system is said to be reconfigurable if it can assume several architectural configurations, such as pipeline or MIMD (Multiple Instruction Stream Multiple Data Stream), each of which is characterized by its own topology of interconnections. In the applications of a multicomputer system for parallel processing of various algorithms it is desirable to have the power of reconfiguration. There are three types of functional units attached to C-bus that comprise CUENET: a master computer, several slave computers, and network memory units (NMU) which are shared memory

banks. The master computer is responsible for the coordination of all other computers of CUENET, the computational tasks required by the end user are carried out by the slave processors. This master slave approach is intended to simplify the complexity of the CUENET operating system.

A simulation model was developed to examine the different alternatives in the engineering design of C-bus. This model was implemented using the GPSS simulation language. Upon examing the simulation results and our projected needs, we found it possible to use a general purpose microprocessor with appropriate hardware extensions as a central controller for C-bus.

In the implementation of our network two kinds of boards were designed and developed: (a) a circuit board containing the hardware extensions required for the C-bus controller, (b) one circuit board for each interface between an off the shelf microcomputer and C-bus. In the experimental prototype of CUENET there is one master, three slaves, and one NMU connected to a C-bus. C-bus employs twisted pairs of wires for signal transmission and is 50 feet long in its present form.

There are two major potential applications for CUENET. It is useful for parallel processing of compute bound algorithms under certain conditions, and as a local area network.

To my parents, Sidney and Ida, for their financial support without which this thesis would not have been possible.

To Miss Johanne Sebek whose moral support and patient devotion to my studies were a large encouragement.

ACKNOWLEDGEMENTS

I am greatly indebted to my supervisor Dr. T. Radhakrishnan. His personal interest in my progress has made my stay at Concordia University a very enjoyable experience. I appreciate the effort he has taken to reduce the bureaucracy that is incurred with a hardware project of this nature. His assistance, advice, and encouragement were an asset throughout the entire project.

I would like to thank Dr. Terry Fancott for his support, help, and the information he made available to me upon request.

I also wish to thank Mr. J. Blaison for the knowledge I gained from observing him and his patience with my questions.

Financial support for my graduate studies has also been providied by the Governement of Canada through the National Sciences and Engineering Research Council.

Mr. Chary Tamirisa spent many hours with me during the implementation phase of the project. The software he developed was instrumental in the testing and debugging of C-bus.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

FIGURES

TABLES

TABLES

CHAPTER I

INTRODUCTION

Interest in multiprocessor systems is not a new phenomenon. Computer designers have been actively working in this field for a variety of reasons [Adams 78, Clark 78, Satya 80, Lam 82, and Thurbl 79]. Four basic motivations for the continued development of multiprocessor systems [Enslo 77] are:

(1) the need for higher throughput of data than what is obtainable from a uniprocessor system.

(2) flexibility to expand, or shrink, the system in meeting dynamic work load requirements.

(3) reliability.

(4) system availability.

Each attempt to design and develop a multiprocessor system to date has focused on one or more of these four objectives [Kuck 77]. The decreasing cost of LSI components, such as processors and memories, has created further interest and new directions in multiprocessor architectures [Arulp 80, Kober 77, and Jones 80]. A single low cost microprocessor of today possesses substantial functionalities; but is limited with respect to throughput capabilities. The main objective of the design in the modular minicomputer project [Arulp 80], for example, has been to increase the data throughput of the system by using multiple microprocessors.

1.1 Multiprocessors and Multicomputers

While multiprocessors and multicomputers each possess multiple processing units (D-units), as defined by Baer [Baer 80], their hardware configurations are different with respect to the sharing of resources among the D-units [Enslo 77]. A multiprocessor is a single computer that contains multiple processing units which share all the system memory and input output devices. On the contrary, multicomputers are a set of separate computers which have direct connections between them. The interaction between D-units in a multiprocessor is usually at the data element level rather than through the transfer of a complete data set, or a message, which is the case with multicomputers. In this thesis we treat the terms multicomputer, parallel processor, and distributed computer as synonymous.

The major aspects of the design of a multiprocessor, or multicomputer system, and its application lie in the design or selection of the following:

(1) A set of processors and memories that are interconnected. They may be homogeneous that is functionally identical, or heterogeneous.

(2) An interconnection network (ICN) that physically connects the different subsystems [Siegel 79].

(3) Division of a "large" process into tasks or computational units that can be executed concurrently on the hardware structure defined by (1) and (2) above.

(4) An operating system that will coordinate the execution of different tasks, allocate tasks to processors, and will manage other processes.

A wide variety of choices is available to a designer with respect to the above four categories. Thus the design of a multiprocessor system, and hence its success, is much more challenging than the design of a uniprocessor system [Karta 82]. As noted by (3) above, it is not adequate to design and construct a multiprocessor system; we need to be able to use it effectively. Jones and Schwarz observe [Jones 80] that understanding parallel solutions is generally a more complex task than the design of their sequential counterparts. There have been several attempts in the past to design application specific multiprocessor systems. The ILLIAC IV computer designed for picture processing applications [Barne 68], and the KENSUR project at IRISA [Andre 80], France, devoted to translation of programming languages, are two examples.

A sequential algorithm can be split, or decomposed, into subprocesses which can be executed concurrently on separate computers. This collection of subprocesses, which we call a parallel algorithm, will collectively attempt to solve the same problem as its sequential counterpart. A given sequential algorithm may be decomposed in several different ways, each of which will require a specific type of interconnection structure between the processors. The problem of arriving at an optimal decomposition for a given sequential algorithm, in general, is far from solved [Enslo 77]. This indicates that there is a need for a "research tool" upon which a proposed decomposition can be

executed and evaluated. Such a tool is indispensable for experimental research on parallel processing [Jones 80]. A "research tool" for testing parallel algorithms will be comprised of a combination of both hardware and software as shown in Fig. 1.1. Such a tool must allow the researcher to specify different interconnection topologies among the processors in order to evaluate various decompositions. The ability to configure different topologies can be achieved either through hardware, software, or a profitable combination of both.

Some time ago Flynn [Flynn 66] classified parallel computers into four categories:

(1) SISD or single instruction stream and single data stream.

(2) SIMD or single instruction stream and multiple data stream.

(3) MISD or multiple instruction stream and single data stream.

(4) MIMD or multiple instruction stream and multiple data stream.

In this context, a stream means a sequence of entities (data or instructions) manipulated by the processors. Parallel computers may also be classified according to the degree of coupling that exists among the computers [Fulle 78]. A loosely coupled parallel computer will be comprised of independent processors that send and receive limited commands or data between themselves. Memory sharing is not practical in this type of system where communications usually take place through low speed buses. In tightly

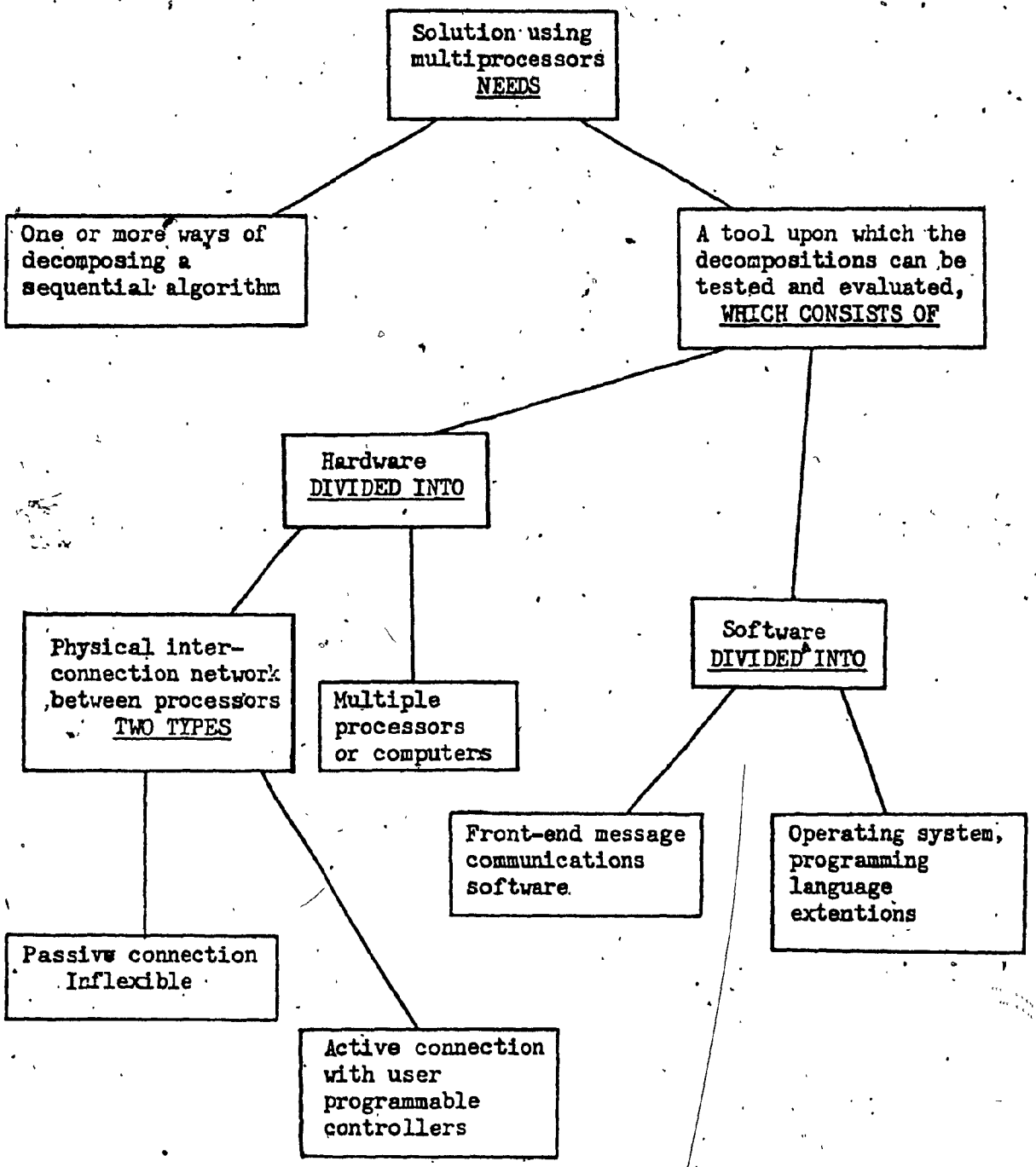

Solution using multiprocessors NEEDS
One or more ways of decomposing a sequential algorithm
A tool upon which the decompositions can be tested and evaluated, WHICH CONSISTS OF
Hardware DIVIDED INTO
Physical inter-connection network between processors TWO TYPES
Multiple processors or computers
Software DIVIDED INTO
Front-end message communications software
Operating system, programming language extentions
Passive connection Inflexible
Active connection with user programmable controllers

Figure 1.1

coupled systems intercomputer communications take place frequently over one or more high speed buses, or common memories [Hirsc 79]. Multiprocessors that have an SIMD architecture will have a central control unit broadcasting instructions to a set of D-units. In this type of structure the processing units are considered to be tightly coupled and are usually located within a single cabinet. Multiprocessors that are classified by Flynn as MIMD type architectures are generally multicomputers. The individual computers that are grouped to form a multicomputer may be at short distances from each other or physically distributed over a long distance. Multicomputer systems that are localized within an office, a room, or a rack have been proposed to support a wide variety of intercomputer communication topologies such as a pipeline [Stone 75], a tree structure [Buchb 79], a cube [Hayne 82, and Adams82], and hybrid structures [Andre 80]. These multicomputer systems are considered to be loosely coupled architectures. Distributed multicomputer systems are further subdivided into short and long haul networks. Short haul networks are intended for communications within a single building while long haul networks are capable of transmissions over distances of thousands of miles. Two major short haul networks, also called local area networks (LAN), that have been proposed are Ethernet and Wangnet [Techn 82].

1.2 Reconfigurability and Decomposition

A computer system is said to be reconfigurable if it may assume several architectural configurations each of which is characterized by its own topology of activated interconnections between computers [Siegel 79]. Vick et al consider reconfigurability as one of three approaches to "adaptable architectures"; the other two being microprogrammability and dynamic architectures [Vick 80]. The primary objective of reconfigurable systems is to change the system architecture so as to match it with the nature of the problem being solved.

Developments in LSI technology and interconnection structures have made it possible to reconfigure a system via software [Karta 79]. With respect to reconfigurability there are two broad classes:

(a) Dynamically reconfigurable - the computer architecture will change dynamically during the execution of a process depending upon the specifications of the software designer.

(b) Statically reconfigurable - the architectural configuration is set at the start of a process and will remain fixed until the process terminates.

For the purposes of our discussion we will be concerned only with the static type of reconfigurability.

When an architecture is well matched to a problem, throughput of the data stream is increased, and unwarranted system overhead can be reduced. For instance, an array type architecture can be reconfigured to reduce the overall dummy

time in the pipeline, and so forth.

System parameters that can be changed for the purposes of reconfiguration may be grouped into two categories:

(a) Intra processor parameters or ISP (Instruction Set Processor) level parameters [Bell 71].

(b) Inter processor parameters or PMS (Processor, Memory, Switch) level parameters.

Adaptable architectures of [Vick 80, and Karta 79] that change ISP level parameters such as word length, instruction set and the like would fall into the former class. Reconfiguration at the level of processors, memory modules, and their interconnections would fall into the latter class. By reconfigurability in this thesis, we mean PMS level reconfigurability.

A basic prerequisite for reconfigurability is a flexible interconnection network, ICN, between the different PMS components. We are aware of two types of interconnections between components:

(a) physical interconnections provided by the ICN,

(b) logical interconnections controlled by the operating system.

A logical interconnection from component i to component j cannot exist without a physical path, direct or indirect, from i to j. Similarly although a physical connection from i to j exists, the logical connection from i to j may be

disallowed by the operating system software. Efficient realization of logical interconnections is an important design factor in reconfigurable architectures. We may not achieve an efficient logical interconnection without a well designed physical interconnection system.

An important feature of any ICN is the existence of a "saturation limit" or a point of diminishing return with respect to the number of processors employed. The ideal increase in processing speed obtainable with a multicomputer is called linear speedup [Jones 80]. Ideally, in a multicomputer that is comprised of n independent computers, computations can be performed at most n times as fast as on one sequential computer. Unfortunately linear speed up is not achieved in present multicomputers because of overhead generated in the control of parallel algorithms due to task synchronization and data transfers between computers. The total system degradation due to these factors is directly affected by the physical nature and topology of the ICN chosen. The amount of overhead will increase as the number of computers is increased until the saturation point of the ICN is reached. At this point the addition of computers does not result in a significant increase in the speed up of a parallel algorithm Fig. 1.2.

Consider the <u>division</u> of a process P into modules, or <u>tasks</u> {t1,t2,...tn}, each of which can be performed concurrently on a multicomputer architecture. These tasks

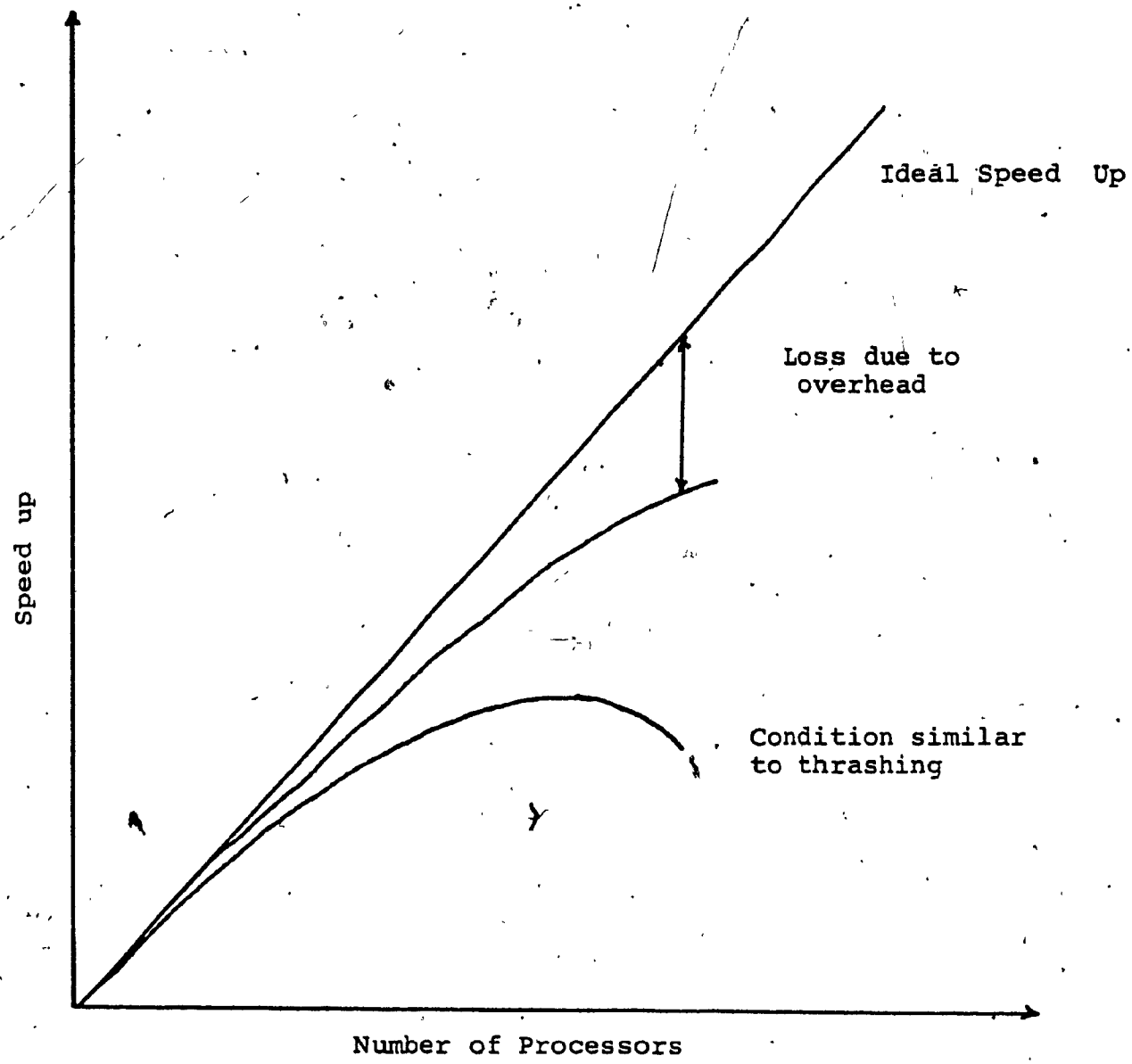


Figure 1.2

Performance of a process with multiple processors

are also referred to as blocks [Lecou 81]. In general each task ti will have some precedence constraints with respect to other tasks. The tasks of a division may be simple, as in floating point multiplication, or as complex as a software process consisting of an arbitrarily long sequence of instructions [Jones 80].

It is possible that a given process may be divided in more than one way and a particular division may be more suitable to a particular architecture than others. Thus, we use the term decomposition D(d,a) to denote the association of a division (d), with a specific architectural configuration (a). There are many different divisions possible for a given sequential process. Even though we may finally wish to choose one division, we still have a large number of possible decompositions to consider.

As an example consider the division of a floating point multiply instruction into tasks such as exponent adjust, mantissa multiply, and post normalization. This division of single machine instruction, is said to be of a "low granularity" or small "grain size" [Jones 80]. On the other hand division of the character recognition process [Vanke 77], into tasks such as noise elimination, which is a division at the level of a software process, is said to be of a "high granularity" or large "grain size". We can get a reasonable idea about the granularity of a decomposition if we examine each task of the corresponding division. We can

measure, or estimate, the average amount of time a task will spend on a unit of data before the data unit is passed onto another task in the system, and this task starts execution on a new unit of data. The grain size of any task is in direct proportion to the execution time mentioned above, but it is difficult to assign an exact cut off point between small and large grain sizes. The granularity of a particular decomposition will be determined by the minimum of the grain sizes of the tasks that make up the division associated with that decomposition. In this thesis we will be primarily interested in the decompositions of high granularity.

1.3 General Purpose ICN

Clearly, the execution time associated with a particular task will be dependent on the data set, the characteristics of the processor on which the task will execute, and the properties of the division itself. For a specific program one could estimate, or measure, the execution time of various tasks which are important to the evaluation process. When the granularity of a decomposition is high, each task is associated with complex state descriptions and communication patterns with other tasks in the system.

We feel that in order to aid the current research in the area of algorithm decomposition for parallel processing

a general purpose reconfigurable multicomputer would be helpful. If such a multicomputer could be produced at a reasonable cost then it would become feasible for many research centers to engage in experimental research. The major application for a general purpose reconfigurable multicomputer will be to provide a medium for testing the decomposition of an algorithm for parallel processing. Presently, researchers are attempting to decompose algorithms that are CPU bound problems, and therefore require a large amount of computing. An algorithm is decomposed for parallel processing with the intent of increasing the throughput of data or reducing the execution time. The actual speed up attained will be a function of the manner in which the algorithm itself is decomposed and the interactions between the decomposed components. An important factor that determines the suitability of a particular decomposition to the capabilities of an ICN is the ratio of interprocessor communication time to the local processing time for each microcomputer of the multiprocessor. Since obtaining an optimal decomposition is still an open research question we see that such a multiprocessor can be used for experimental research as depicted in Fig. 1.3. Some problem areas that are investigated for parallel processing include real time processing of speech signals, combinatoric computing, sorting, character recognition, distributed compilers [Andre 80], office automation and distributed databases

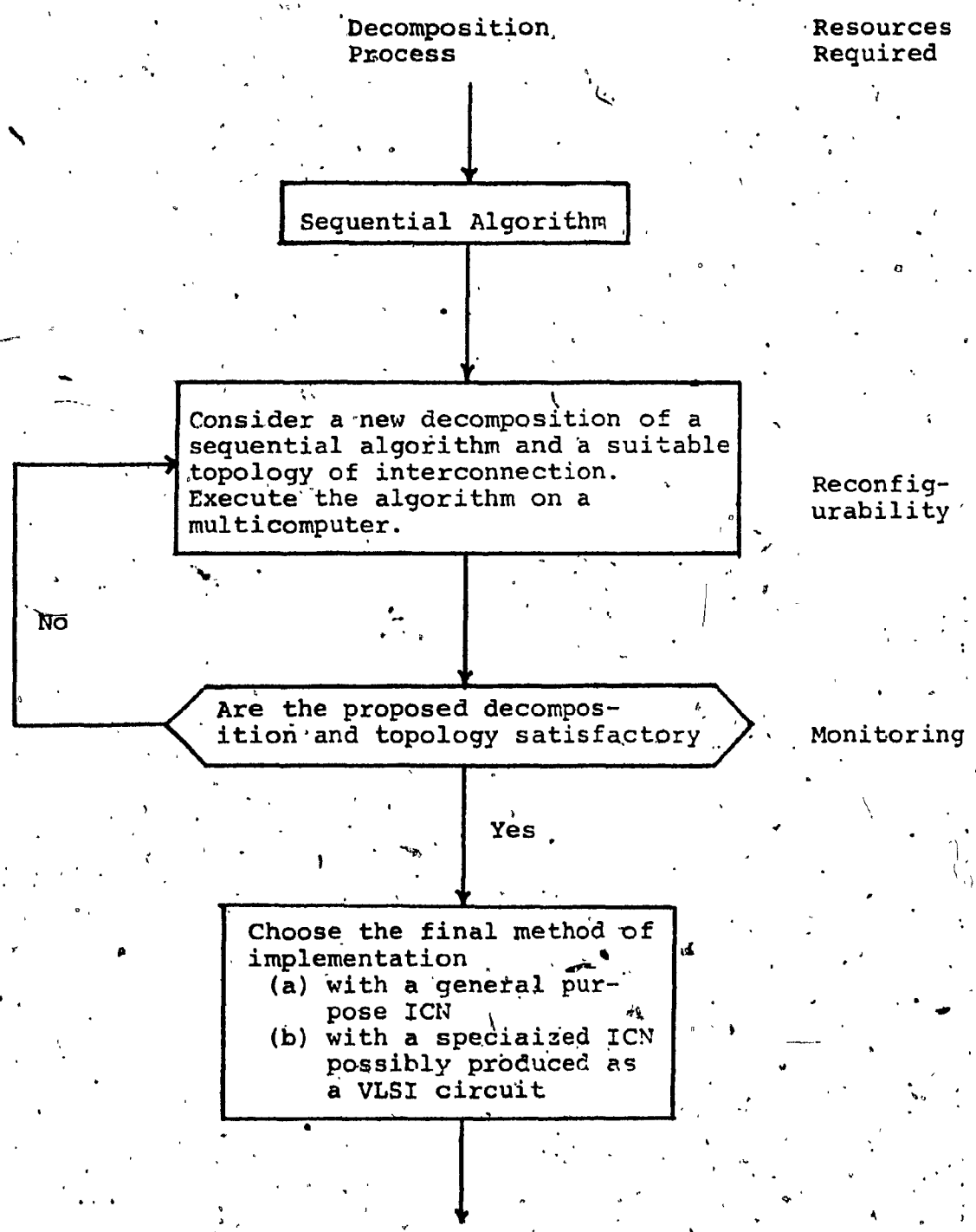

Decomposition Process
Resources Required
Sequential Algorithm
Consider a new decomposition of a sequential algorithm and a suitable topology of interconnection. Execute the algorithm on a multicomputer.
Reconfig-urability
No
Are the proposed decomposition and topology satisfactory
Monitoring
Yes
Choose the final method of implementation
(a) with a general purpose ICN
(b) with a speciaized ICN possibly produced as a VLSI circuit

Figure 1.3

based on microcomputers.

1.4 Outline

In Chapter II we will examine some of the more popular interprocessor communication topologies. Several of the multiprocessor and multicomputer systems are described. In addition some standard interprocessor communication standards are examined. C-bus, an ICN designed to support a general purpose reconfigurable multicomputer system, is proposed in Chapter III. The features, benefits, and limitations of C-bus are discussed and contrasted with the multicomputer systems described in Chapter II. In Chapter IV a simulation is described in order to determine the capabilities and the saturation points of C-bus under different load conditions. A simulation model is described and the simulation results are presented. The details of our implementation of C-bus are given in Chapter V. Our design approach is presented along with block diagrams and the timing requirements of the hardware. The software drivers for C-bus are also explained. Chapter VI is concerned with the intended applications of a multiple computer based on the C-bus. The classes of parallel algorithms that are well suited for this architecture are enumerated. The possible applications and future hardware developments envisioned for C-bus are outlined. Finally, the software development required to support a C-bus based multicomputer is described.

CHAPTER II

AN OVERVIEW OF INTERCONNECTION MECHANISMS AND MULTICOMPUTER SYSTEMS

A wide variety of multicomputers have been proposed by researchers and constructed by the computer industry. The computers that comprise a multicomputer system may be separated by large distances or contained in a single integrated circuit. Datapac and Arpanet are two networks in operation today that support transmissions over thousands of miles [Weitz 80]. These networks employ a packet switched, message based, bit serial transmission protocol. Local area networks (LAN) have been announced by many computer manufacturers in the past year [Techn 82]. These communication systems are designed to operate within one building, office, or a single room. For example Ethernet [Metcal 76] is capable of transferring information between two computers located within one building and Micronet [Witti 78] will provide communications between computer units contained on a set of racks within a single room. These networks employ a message based protocol and the transmission mode may be either bit, or byte, serial. Decnet has been designed by Digital Corporation to support interprocessor communications between DEC mainframes.

Multicomputers, or multiprocessors, that are located a short distance from each other, usually within the same cabinet, are generally tightly coupled architectures and use

some form of shared memory system. Several prototypes for multiprocessors have been built at various research centers such as Cm*, C.mmp, MP/C, and U* [Wulf 72, Swen 77, Arden 82 and Civer 82]. In addition, several large multiprocessors, also called supersystems [Swert 82], are available like the CYBER 200, STARAN, and Burroughs Scientific Processor [Linco 82, and Kuck 82]. In fact, several of these supersystems have found applications in military defence systems [Berg 72 and Batch 82]. R. Arnold et al have designed a modular supersystem architecture where basic hardware modules can be assembled to provide various configurations dependent upon the application [Arnol 82]. At Rockefeller University a reconfigurable laboratory instrumentation system, to be used for monitoring experiments, has been constructed which allows for the definition of the functioning of the system based upon the needs of the current experiment [Silve 82]. The current advances in VLSI technology will soon make it possible to create integrated circuits that contain many individual processors [Mead 80, and Fairb 82]. Consequently many researchers have proposed architectures comprised of many processors with complex interconnection structures [Hayne 82, Gottl 82, and Kuny 82]. A specialized structure for performing matrix arithmetic was proposed by [Ahmed 82] and a Fast Fourier Transform algorithm has been suggested as an application. The CHIP processor, designed at Stanford University, is comprised of an array of processing elements

connected by a switch lattice which is configurable under software control [Snyde 82].

A processor interconnection scheme is an essential part of a multiprocessor or multicomputer system [Akkoy 74]. An ICN may be classified according to its topology or the structure of its implementation. The entire spectrum of ICN's and multicomputers is beyond the scope of this thesis. In the following sections we will discuss the various ICN's and multicomputers that are most relevant to our research.

2.1 Interconnection Strategies

The choice of an ICN topology or implementation is one of the most crucial design considerations for a multicomputer. Each topology is best suited to some specific applications rather than others. In addition, an ICN will impose many restrictions upon the final product. The choice of an ICN will be based upon many factors such as cost, reliability, response time, speed, throughput capacity, and modularity. In this section we will introduce some of the more popular ICN mechanisms.

Many multiprocessors that are tightly coupled rely on a shared memory scheme and can be implemented using a number of methods. In some multiprocessors one single bank of primary memory is present that can be accessed by various processors. This memory is used to hold data but each processor will have its own private memory for storing

programs. A standard approach is to use a common bus through which all system resources are shared. Although this is a simple interconnection structure, if it fails the entire system cannot function and contention for the use of the bus can be a problem. In fact, the speed of the multiprocessor will be limited by the bandwidth of this single path. As a possible solution to the contention problem it has been suggested that multiple buses are possible. As the number of buses increases the complexity of information routing via hardware switching is also increased. In a crossbar matrix, or switch, there is a separate bus for each memory unit. All arbitration and routing is the responsibility of the ICN. The switches needed for this type of ICN are complex, costly and large. However, the crossbar matrix can support simultaneous information transfers between different processors and memory units.

In a loosely coupled multicomputer system the demands made on the ICN by each computer are less frequent but the size of a transfer is generally larger than for tightly coupled systems. The simplest ICN that has been constructed to date uses a single time shared bus to support interprocessor communications. Information is usually sent in the form of messages of varying length. All modules that can access the bus will have a unique ID code which is used for bus arbitration. Control of this type of ICN can be either centralized, as in IEEE 488 [Gilbe 82] which uses a

central controller, or distributed as with Micronet [Witti 80]. For centralized systems the bus controller may be a standard processor or a special purpose controller. This interconnect structure allows for the easy removal or addition of computers to the network. The cost of a time shared bus is low but it will become a critical component which will degrade the overall system performance as the demands on the bus increase. This can be overcome by adding additional buses but the complexity of the ICN is increased.

In the United Kingdom there has been much interest in a high speed unidirectional communication channel arranged in a closed loop. Messages are sent between computers by placing them onto the loop where they will circulate around the ring until the destination computer is reached. Messages are either removed from the ring by the destination computer or allowed to continue circulating around the ring until it reaches its sender. This structure has been used to transmit only digital information but no message routing is required and many messages may be on the loop at a time which allows for a high throughput. There are three different protocols that have been proposed for ring based ICN's.

Farmer and Newhall [Weitz 80] constructed a loop structured ICN where a control token circulates around the ring in a round robin fashion. A computer station that wishes to transmit information will wait until it possesses

the control token. The computer will then hold the token until its message has been sent. Under this protocol only one message can be transmitted at a time which will limit the throughput. In addition, when a station wishes to transmit it will be delayed until the control token arrives even when no other station is using the ring.

In the Pierce type loop, space on the ring is divided into fixed size slots into which information can be placed. Messages are sent by slicing them into packets which can be placed into empty slots on the loop as they pass. Each slot will contain some control information which will indicate if it is occupied. In the delay insert model variable rather than fixed packet sizes are allowed. A message is inserted into the loop by intercepting all information passing on the loop long enough to place the desired message onto the loop. All the information that was intercepted is then placed onto the ring following the new message. In this manner all the information circulating on the ring is delayed by the length of the inserted message [Weitz 80].

2.2 Application Specific Architectures

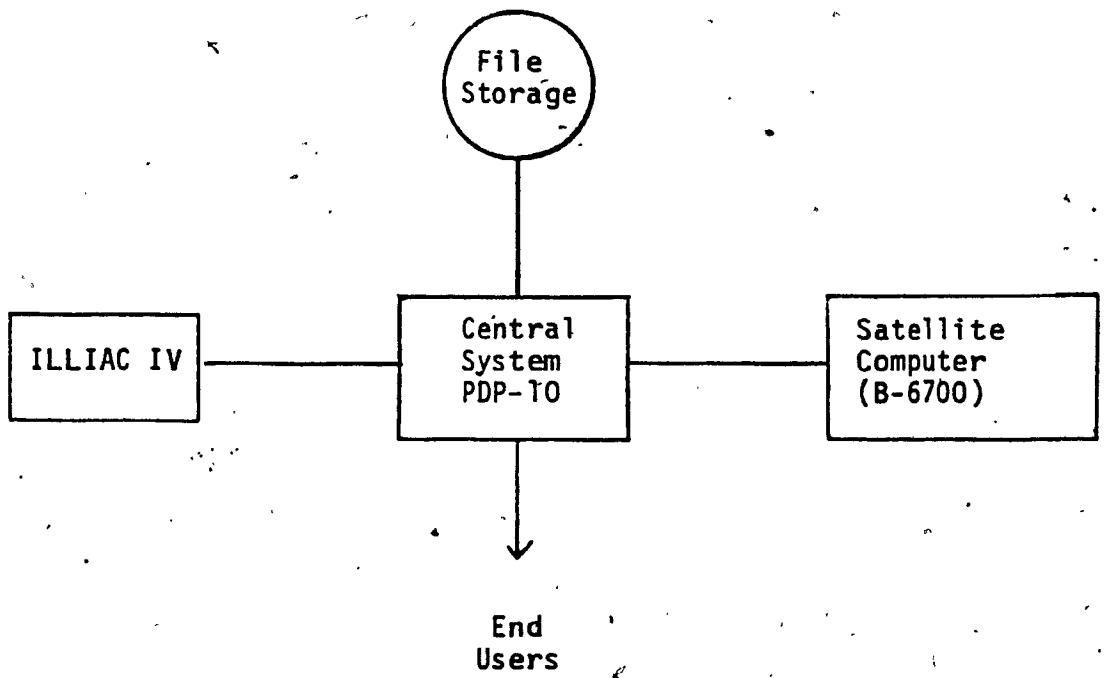
Early attempts in developing multiprocessors were aimed at specific problems. Consequently, these multiprocessors are well matched to a specific problem area. They are rigid in structure and useful only in a limited context [Thurb2 79]. For example, Illiac IV is a multiprocessor

system designed primarily for image processing. The problem of handling large volumes of data arises in many data base applications and data base machines have been proposed for this purpose [Ozkar 77].

Illiac IV was intended to solve problems that are based on matrices of data and have a potential for a high degree of parallelism. Problems which fall in this category are the solutions of differential equations, matrix operations, and weather data processing. To this date Illiac IV has been used to solve two-dimensional aerodynamic flow equations, to simulate weather and climate prediction models, for signal processing, and linear programming [Siewi 82].

Illiac IV is an SIMD architecture and operates as an expensive and powerful attatchment to a central computer system [Baer 80]. The global configuration is given in Fig. 2.1. All communications for input and output are handled by the central computer while the operating system, compilers and assembler are resident in the satellite computer. Illiac IV itself contains one control unit, 64 arithmetic units, and 64 memory modules. All of the arithmetic units operate synchronously under the control unit. The control unit is also a small computer capable of performing scalar operations in addition to controlling the vector operations of the other 64 arithmetic units.

There are three major internal buses in Illiac IV; a



ILLIAC IV

(From Baer 80)

Figure 2.1

unidirectional link between the control and arithmetic units through which the control unit broadcasts operands, a unidirectional link between the control unit and memory units used to fetch instructions and operands, and a routing network allowing the arithmetic units to transfer information among themselves. The routing network allows each arithmetic unit to communicate with its four nearest neighbours which are labelled north, south, west, and east. This will facilitate high speed data sharing.

Data base machines have been proposed using SIMD, MIMD, and back end type processors. In each case specialized hardware was produced to support the standard data base management functions which reduced the complexity of the resident software. A data base machine using an MIMD architecture called DIRECT [Dewit 79] has been constructed where processing modules were connected to an array of shared memory units through a cross bar switch. Data base machines using a back-end approach usually employ a mini-computer to perform functions such as access validation, storage management, concurrency control, and input output control in order to support a large front end main frame.

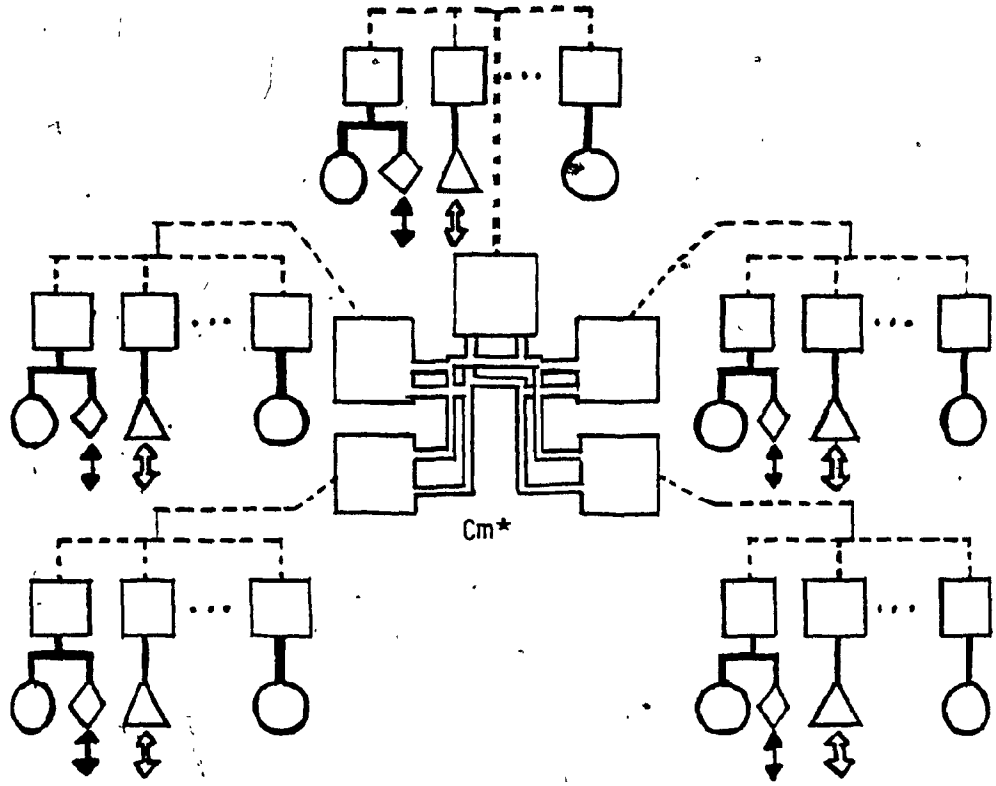
2.3 Multi-Microcomputers

The first major attempt to design a large multicomputer was made by Carnegie-Mellon University in the construction

of C.mmp [Wulf 72]. The C.mmp multicomputer system used a 16 x 16 cross bar switch to provide processor interaction through a set of shared memory units. The computers used were slightly modified PDP-11's. Besides using the cross bar switch, each computer can send control information to other computers over a time shared bus called the "interbus". The interbus is used by one computer to interrupt another computer or as a global time source. The Hydra operating system was then written for C.mmp. However, due to the cost and complexity of the cross bar switch C.mmp was never expanded past 16 computers. As a result of their experience with C.mmp the researchers at Carnegie Mellon designed Cm* [Swan 77].

Cm* is a multiprocessor system which consists of a set of closely coupled LSI 11 microcomputers Fig. 2.2. The communication structure between the LSI-11's is fixed in the form of a hierarchy. Sets of LSI 11's are grouped into clusters and within a cluster they share a single time shared bus called a Map bus. Into each LSI-11 a unit called an S-Local (local switch) is inserted to give it access to the map bus. The S-local will route the memory requests of the local LSI-11 to either the local storage or onto the map bus and is also responsible for answering requests from any of the external processors.

All clusters are linked together by another time shared bus called an intercluster bus. Each cluster is connected



Legend

Intercluster Bus
Map Bus
PDP-11 Bus
DA Links
SLU to Host

KMAP
CM
Disk
SLU
DA Link

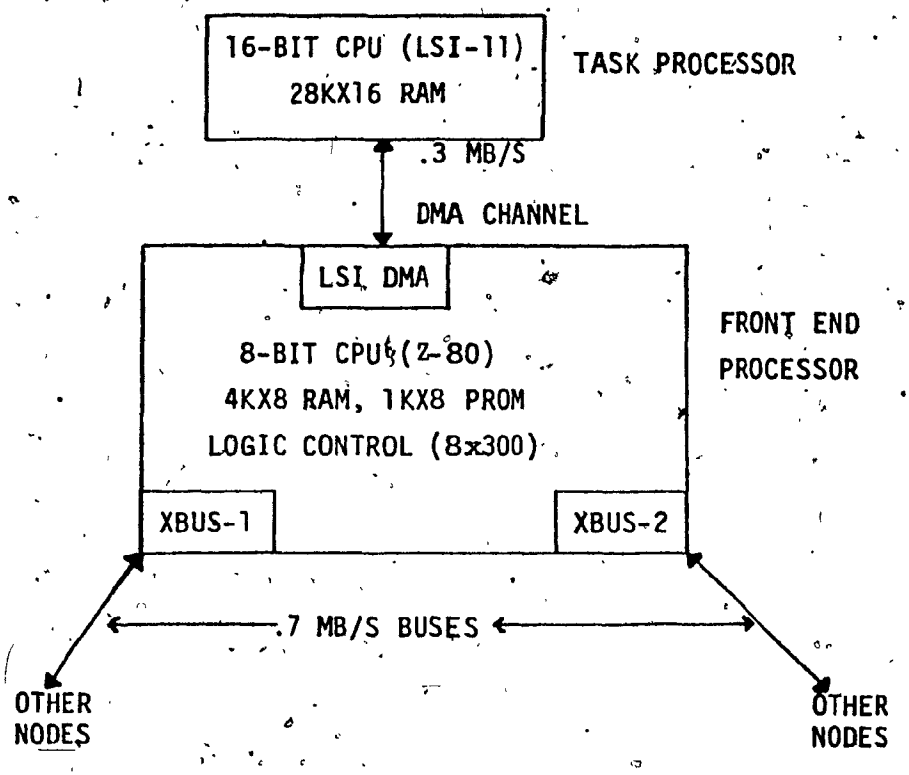
Cm* (Five Clusters)
(From Swan 77)

Figure 2.2

to the intercluster bus through a special purpose processor called a kmap. The kmap is a special purpose microprogram controlled bit slice processor. It can manage eight active requests at one time. In addition the microcode of the kmap is designed to support various serialization primitives required for distributed program control and operating system functions. In effect these primitives will appear as an extension to the LSI-11's instruction set.

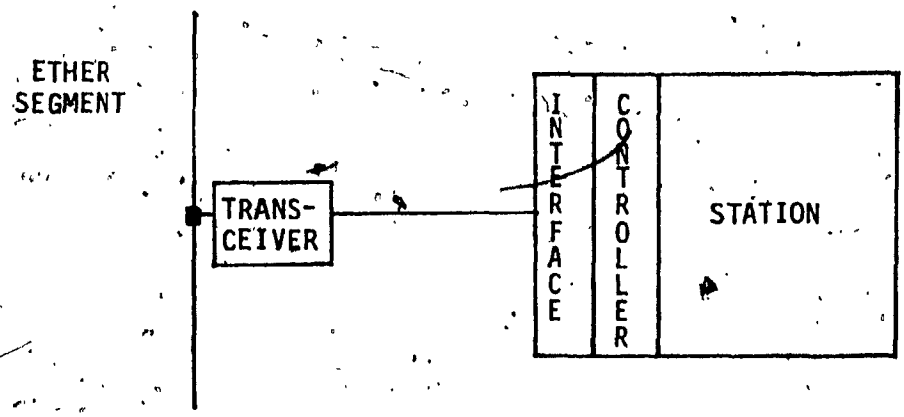
The LSI-11's can access any data element independent of where it is stored. The communication system for Cm* is elaborate and expensive. Each memory access made by any of the LSI-11s is monitored to determine if the concerned address is local to that computer, its cluster, or outside of its cluster. A processor is halted if the information must be obtained from elsewhere in the network. This monitoring will generate a measurable amount of overhead, which is unnecessary when the information is available locally. Cm* does not allow for the reconfiguration of the interprocessor communication topology and all the computer units are homogeneous.

Micronet is a network of loosely coupled microcomputers intended for general purpose applications [Witti 78]. Fig. 2.3 shows the structure of a Micronet node. Each computer station is comprised of an LSI-11 and a communications front end. The front end contains one general purpose microprocessor, Zilog Z-80, and a special



ONE MICRONET NODE

Figure 2.3



ETHERNET STATION

Figure 2.4

purpose processor signetics 8x300. In addition, the front end cards have three high speed ports, and two external ports, each of which provides access to an external bus and one DMA channel through which the front end communicates with the host computer. A DMA channel is also provided between the memory of the 8x300 and that of the Z-80. The 8x300 processor is responsible for the control of these four transmission channels. The Z-80 processor contains the communications kernel of the distributed operating system and the queues of messages awaiting transmission or transfer to the host processor. The interface units of Micronet are complex and expensive, in fact, they have almost as much processing power as the processor station being connected to the network.

The Micronet is reconfigurable by a set of jumper boards but not under software control. Heterogeneous stations are possible but none have been introduced to date. There is no shared memory among any of the computer stations, all processor interaction is performed through messages. The control of message transmissions is distributed over all the nodes. Messages sent on the Micronet are packet switched and vary in size from 16 to 256 Bytes. A computer station need not be directly connected to the station with which it wishes to communicate. The message will automatically be relayed through other intermediate nodes.

Upon wishing to send a message the sender in Micronet must obtain one of the two communications buses using an asynchronous protocol. Once the bus has been obtained the message is transferred in synchronized bursts. In order to allow for synchronization all front ends contain a crystal controlled clock which are all periodically synchronized. Upon the receipt of a message, if a check sum error is detected the sender will be asked to retransmit.

2.4 Interconnection Standards

The various interconnection schemes that have been introduced by current industry standards or standards organizations, are useful interprocessor communication tools. They can be applied to the construction of multiprocessors but the application of these buses is still left to researchers. The major advantage attributed to these methods is that off-the-shelf integrated circuits are available to support them. This would reduce the cost and development time of a multiprocessor. However, each scheme has a fixed communication structure that cannot be readily changed to meet the needs of specific applications.

Ethernet is a branching broadcast system and is designed to carry digital information between locally distributed computers [Metca 76]. The Ethernet node contains an ethernet controller, an interface unit, and a transceiver (Fig. 2.4). The controller and interface reside

within the host computer and provide access to the Ethernet through a transceiver. The transmission of information occurs in a bit serial fashion over a single coaxial cable.

The control of Ethernet is distributed using a collision detection and recovery strategy based on statistical arbitration. This method is called carrier sense multiple access with collision detection (CSMA/CD). Transmissions wait at the sender until the sender detects that no other packets are being broadcasted. During the transmission of a packet, if interference with other packets is detected due to two or more stations attempting to use the network simultaneously, the transmission will be aborted. After a period of interference free transmission all stations recognize that the Ethernet is in use and the current packet will run until completion. Ethernet controllers in colliding stations will generate random retransmission intervals to avoid repeated collisions.

In the case of many stations, the overhead of this bus arbitration method could become a limiting factor rather than the transmission capabilities of the cable. Any information transmitted through Ethernet is heard by all the stations but only those who recognize the destination address will read the message packet. A broadcast packet is also permitted so that one station may pass information to all other stations by initiating only one message. Ethernet has no mechanism which will permit reconfiguration of the

interconnection topology. Ethernet has been designed to operate in a loosely coupled computer network and it only attempts to maximize the probability of the correct arrival of a message but in no way guarantees the arrival of a message [Metcal 76]. It is left to the software of the sender and receiver to verify the validity of a message.

The cost of an Ethernet interface varies from $3000 to $5000 depending on the type of device that is being attached to the network. In the case that a microcomputer station is to be connected to Ethernet, this cost is considerable.

The IEEE 488 bus standard, or General Purpose Interface Bus, was designed to offer a uniform method for parallel transmission of data [Gilbel 82, and Santo 81]. This standard specifies the types of data and transfer protocols that are used to provide a communications medium. At present the integrated circuits to support this bus standard are commonly available [Willi 79], but interfaces to connect multiple computers are not.

The IEEE 488 bus uses a central controller which performs arbitration of which station will be permitted to send the next message. Once a sender has been selected control of the bus is relinquished by the central controller and the message transfer takes place under control of the sender and receiver. The interface unit required for each station provides three functions: it will prepare the digital code for transmission in the driver and reciever

portions of the interface, encodes and decodes the information sent on the GPIB, and also performs all the control functions required for information transmission as defined by the IEEE standards. The actual transfer of messages between the sender and receiver requires a handshaking on every byte.

The IEEE 488 bus standard has been mainly used for instrumentation control up to this date. There are potential problems that could be encountered by attempting to use this bus for high speed interprocessor message communications. The IEEE 488 bus requires the sender and the receiver to be actively tied to each byte transfer and in addition the sender must wait for an acknowledge signal from the receiver for each byte transfer. This could cause large transmission delays if the receiver is a slow processor. Finally, the address, data, and control lines are all multiplexed over the same set of wires which reduces the number of separate lines required in the transmission cable but causes an appreciable reduction in speed.

The ring based communication network constructed by Cambridge University consists of a set of links between computing stations that form a closed loop [Cambr 80]. Each interstation link is comprised of two twisted pairs and is limited to a short distance. Many message packets, or buckets, are constantly circulating around the ring. A computer station can send information by placing its message

in an empty packet as it passes on the ring. The size of each packet is fixed at 38 bits, but only 16 bits are available to transport information because the remaining 12 bits are used for packet control information.

Each station can only have one message packet circulating on the ring at a time. After a message has been placed in a packet and the destination address is set, it will circulate from station to station around the ring until one station recognizes its own address in the message header destination field. The receiving station will acknowledge the message by marking it as accepted, ignored, rejected, or busy. The message packet will continue to circulate around the ring until it reaches the sender where the packet status is read, and then the packet is marked empty if it has been received successfully.

In addition to computing stations there is a special purpose station called a monitor station. This monitor station will be responsible for removing packets in which errors have occurred and checking for packets that are constantly full because a message has been placed into them but is never removed by the transmitter. These packets are detected by including a monitor pass bit in each packet header which is used to count the number of times a full packet passes the monitor station without being removed by its sender. The monitor station will mark these packets as empty, and control the total number of packets in

circulation on the ring at a time.

The maximum transfer rate for the ring excluding the packet headers, footers and gap bits is 4 M bits/second that are shared among all the stations attached to the ring. The ring structure is inflexible and the delay associated with the delivery of a message can become considerable for a large ring since every message passes through each station.

2.5 Local Area Networks

There have been numerous product announcements in the last year in this area [Techn 82]. Although Local Area Networks (LAN) provide a communication medium between computing stations, the intended applications usually require that most computing stations work independently with only occasional intercomputer transmissions. The major applications area for LAN's has been in linking sophisticated office equipment and personal computers to form the backbone of an automated office. These LAN's are capable of supporting communications within one building.

All of the LAN's have unique architectures and communications protocols but they can be grouped into two broad categories called broadband and baseband. Baseband communication networks can only transmit digital information while broadband systems can support the transmission of voice and video signals in addition to the digital data.

Presently a large debate is underway between various manufacturers about which technology, broadband or baseband, is superior. The producers of LAN's using the broadband technology claim that the baseband technology is limiting because it cannot support video or voice tranmissions and prophesise that it will become obsolete in the near future [Kleel 82]. The manufacturers using baseband technology claim that the broadband supported LAN's are expensive, complicated and not ready yet. The merits of broadband vs baseband systems is beyond the scope of this thesis and will not be discussed further.

The major baseband LAN is Ethernet which uses CSMA/CD and is described in section 2.3. Wangnet [Techn 82] is an example of a broadband topology which has a bandwidth that spans the 10 to 350 megahertz range. Wangnet employs two cables, one for transmission and the other for receiving. The other prominent LAN's are listed in Table 2.1 [Techn 82].

Representative local area networks introduced in 1981

Local area network	Vendor	Maximum data rate, megabits per second	Maximum number of terminals	Maximum distance between terminals, kilometers
Ethernet	Xerox Corp. *	10	1024	2.5
Net/One	Ungermann-Bass Inc. *	4	200/segment	1.2
HYPERchannel	Network Systems Corp. *	50	64	0.5
Z-Net	Zilog Inc. *	0.8	255	2.0
Modway	Modicon Division of Gould Inc. *	1.5	Not available	8
Wang Net	Wang Laboratories Inc. @	12	6500	15
Cable Net	Armdax Corp. @	14	1200	80
Localnet 40	Sytek Inc. @	2.5	Not available	7

* - baseband Technology

@ - broadband Technology

(From Techn 80)

Table 2.1

CHAPTER III

THE ARCHITECTURE OF CUENET

3.1 Design Objectives

C-bus is intended to support a loosely coupled multicomputer, called CUENET, where the interconnection topology is reconfigurable under software control. C-bus will provide a high speed mechanism through which distinct microcomputers can transfer information among themselves in the form of messages. Since the capacity of C-bus will be the major limiting factor for the speed up obtainable in parallel processing, we will attempt to minimize the bus occupancy time for each message by minimizing the overhead of each transfer. In addition, we wish to obtain as large a transfer rate as possible. However, a minimal cost is also a requirement in order that an implementation could be attempted. These two objectives are usually in direct conflict with each other where an increase in transfer rate is usually accompanied by an increase in complexity and cost.

When one is attempting to analyse the cost, or complexity, of an ICN, all observations or measurements must be made relative to the cost and complexity of the computers that will be using the ICN. Naturally our goal was to produce a low cost ICN and as a result the interface logic needed to connect a microcomputer to C-bus could not be logically complex. Secondly, this would make these

interface units more reliable and reduce down time due to hardware failures.

Standard microcomputers do not offer the hardware or software designer any mechanism for restricting access to specific areas of memory. This feature, along with others such as a supervisor mode, with its privileged instructions, and user mode, are present in most larger computers. This poses a problem because the interface of a microcomputer to C-bus will exist as a portion of the memory space and we wish to protect the C-bus interface from invalid accesses by user software. It is quite possible that a user program might intentionally or unintentionally, due to a program crash, damage a process on another computer by sending invalid messages. Therefore the C-bus interface will contain an extra hardware mechanism by which the system software can "lock" the interface from the user programs. In addition, the lock will also make it difficult for any user to sabotage the operations of C-bus which all computers depend upon for their operations.

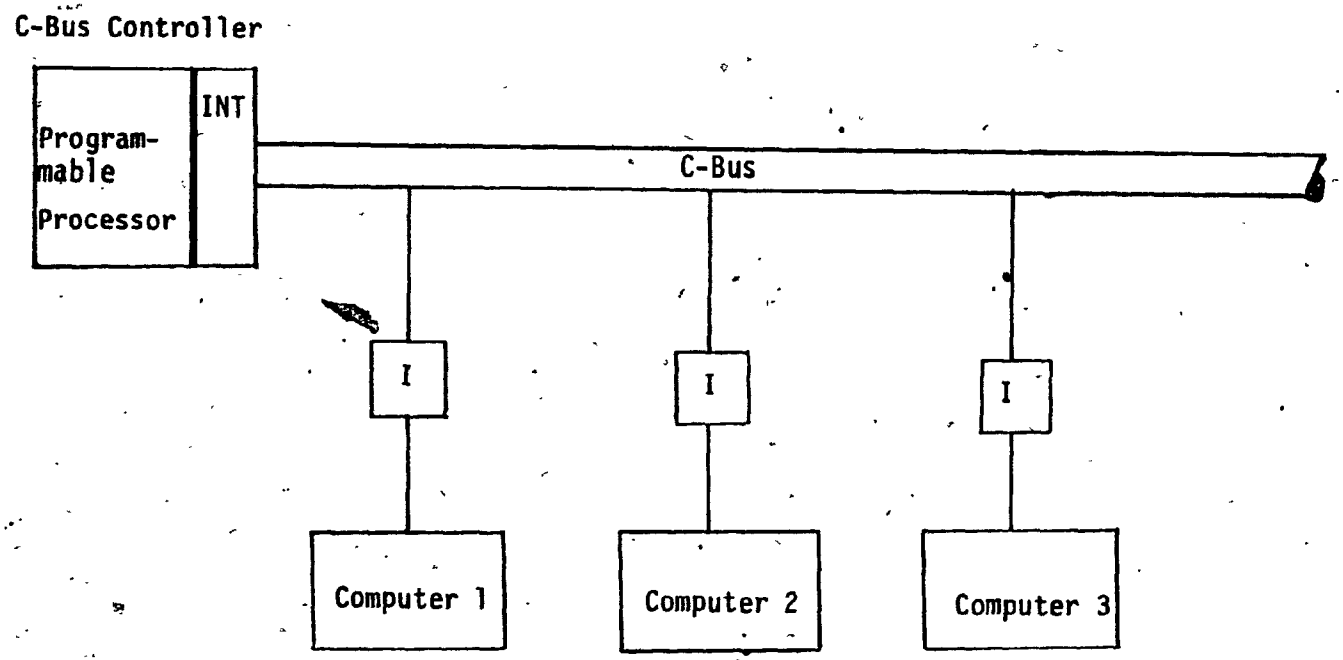
The ICN we have christened C-bus is essentially a time shared bus. This enables the cost and complexity to be low when compared to the cost of standard microcomputers and other ICN's that have been constructed. However, the transfer rate of C-bus will of course be limited but is still fast enough to be competitive with other ICN's and useful in a wide variety of applications. C-bus is

controlled by a special purpose processor, called the C-bus controller, which makes it possible to concentrate the complexity of C-bus in a single unit and reduce the cost and complexity of each C-bus interface. The C-bus interface also contains an access vector and hardware lock. These hardware devices will support reconfiguration of the communication topology between processors and protection of the interface from invalid access attempts.

3.2 Architecture of C-bus

C-bus, a time-shared bus which can provide intercomputer communications, is shown in Fig. 3.1. The C-bus controller functions as a mailman guaranteeing the safe delivery of messages from one computer to another. The logical block diagram of the C-bus controller is given in Fig. 3.2. Essentially a programmable processor is augmented with special purpose hardware that will enable programs running on the processor to control the operations of the C-bus. This is made possible due to the interface selection logic, the data line tranceivers, and the C-bus arbitration and control logic. The real time clock module will be used for time stamping each message processed by the bus controller which is useful for certain error recovery procedures and synchronization control among concurrent processes.

The bus controller will play a crucial role in



I: Bus-to Processor Interface unit

INT: Special purpose hardware added

Figure 3.1

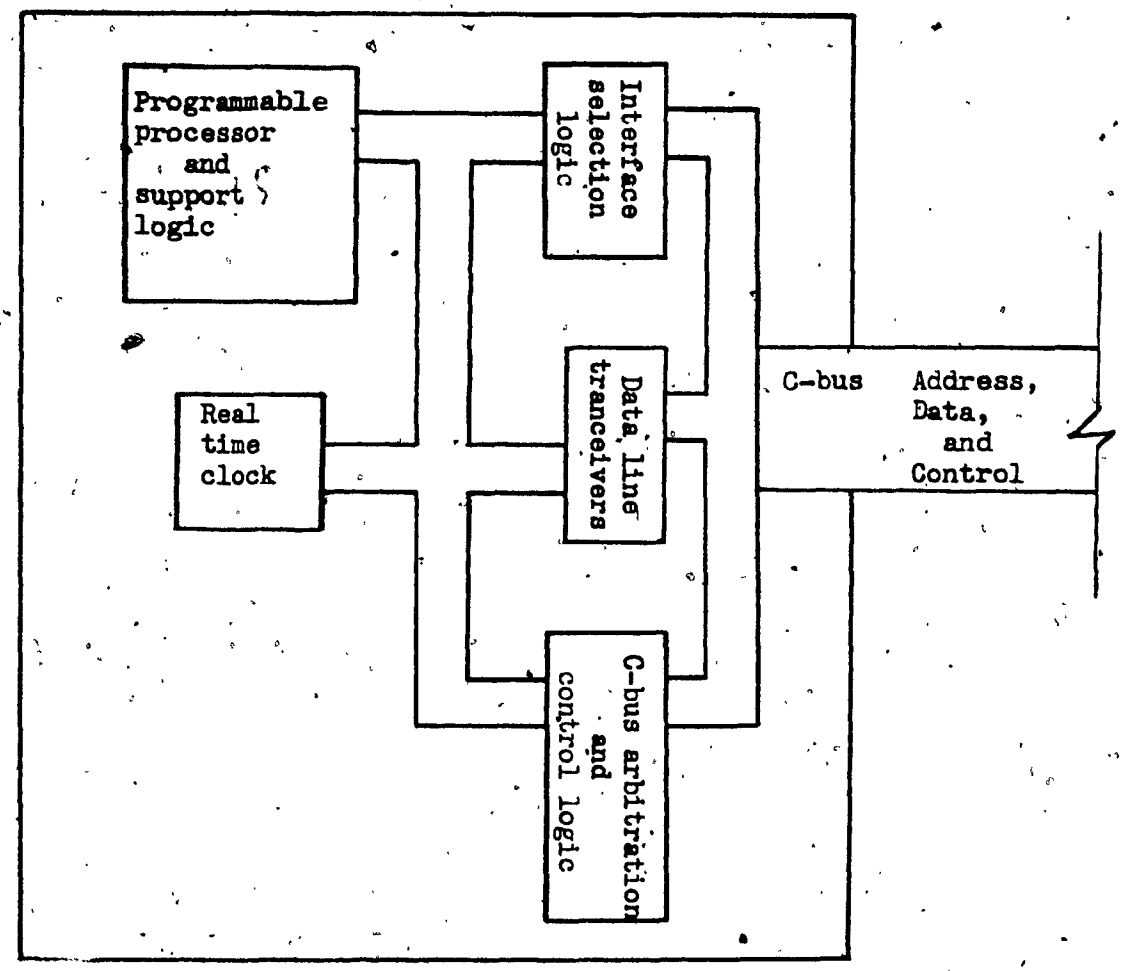


Figure 3.2

C-bus Controller

determining the transfer rate of C-bus. One obvious parameter is the cycle time, or instruction execution speed, of the programmable processor chosen which will be dependent upon the technology and complexity of the processor. Secondly, because of the overhead of a general purpose processor, any control functions that are handled by the other special purpose hardware can be performed much quicker than possible by the general purpose processor. Thus another trade-off exists because as more functions are performed by the additional hardware, the greater the speed of the bus controller, but the complexity of the special purpose hardware circuits is also increased.

The C-bus interface is shown in Fig. 3.3. Messages are sent through the input and output buffers. Contention between the C-bus controller and the host computer for the use of the C-bus interface is resolved by the interface status and control block. The input and output buffers are treated as distinct entities which make it possible for the host computer to deposit a message in its output buffer while the bus controller delivers a message to the input buffer. The daisy chain logic module enables the interface to identify itself to the C-bus controller when the output buffer contains a message to be sent. The parity control hardware will generate or verify a single parity bit for each byte transferred on C-bus. The output of this hardware is monitored by the C-bus controller arbitration and control module in order to identify when a parity error has

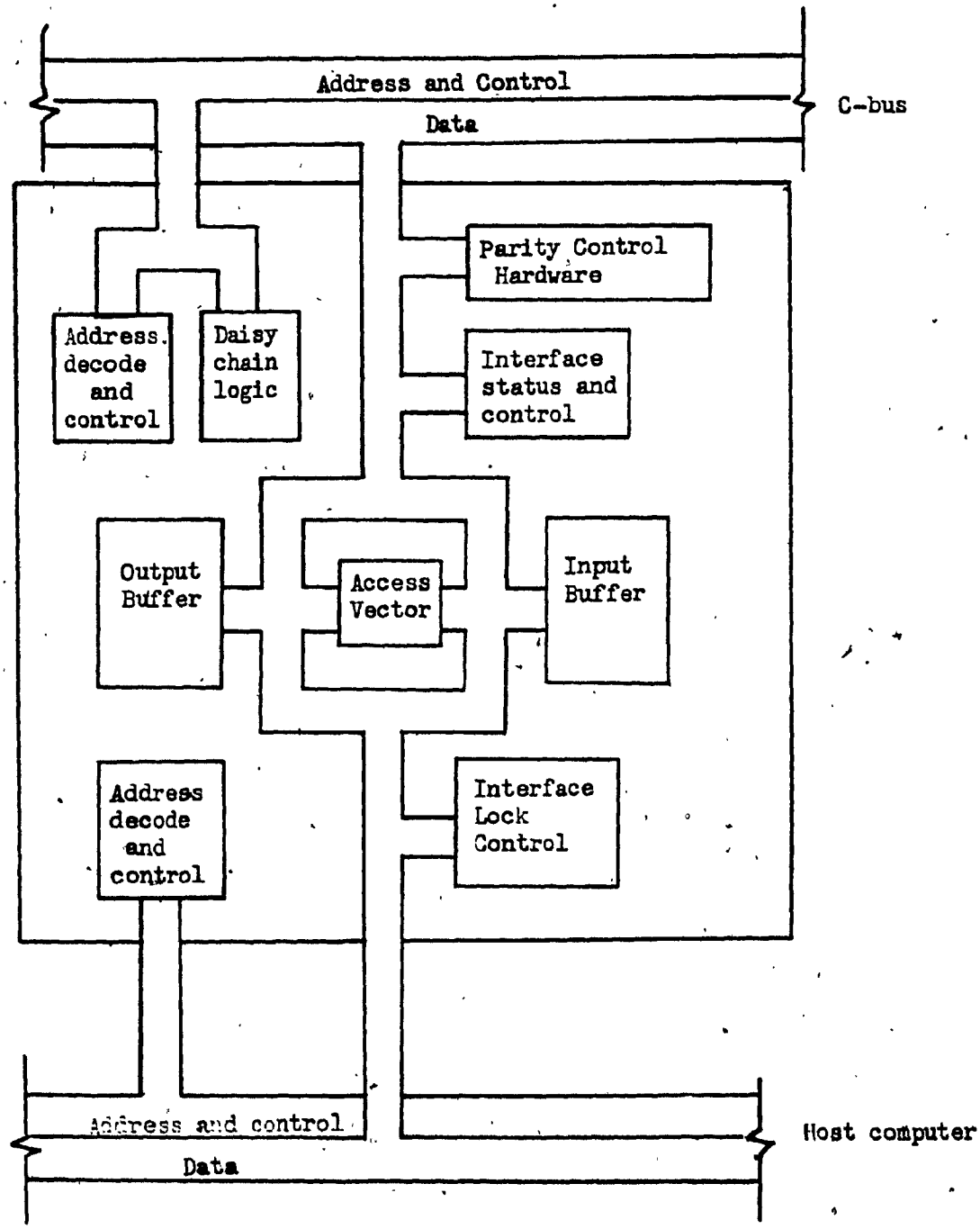


Figure 3.3
C-bus Interface

occurred. The address decoders and control units are used by the C-bus controller and host computers to access the various interface hardware modules.

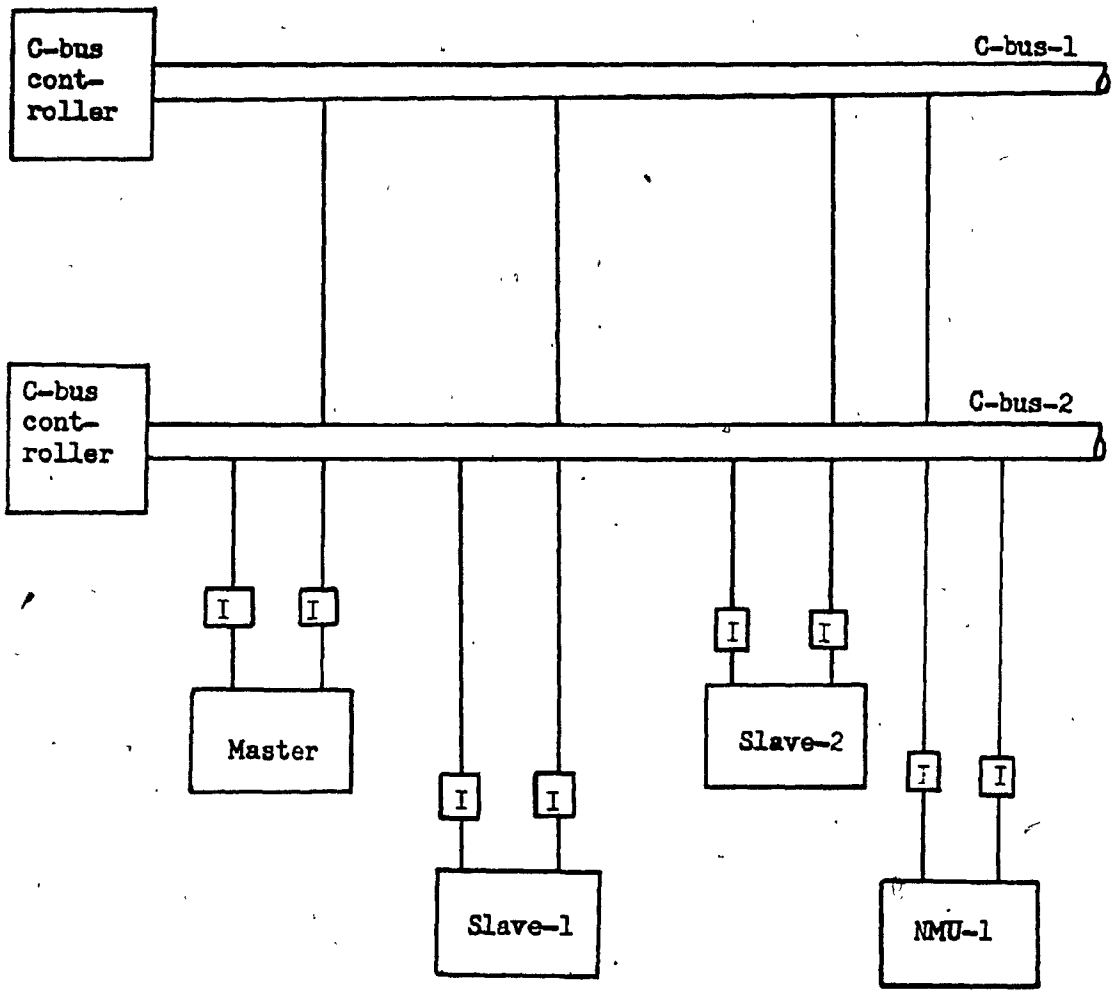
Each C-bus interface contains an access vector. This access vector is essentially a hardware table which can only be read by the host computer, but not modified. One computer connected to C-bus with an authorization from the C-bus controller can initialize the access vectors of the other computers. The information contained in the access vector of a computer will determine the computers with whom that computer can communicate. In this manner one computer can configure the intercomputer communication topology of all the computers attached to C-bus. The access vector will be used in conjunction with the system software to guarantee the integrity of the interconnection topology by disallowing a request for communication with a computer not specified in the access vector.

The interface lock control unit contains two registers called lock and key registers. The contents of the lock register is set to a fixed predetermined value, called a combination, when the interface is constructed. A software process will be allowed to access the interface only if it unlocks the interface, which it may do by writing the combination contained by the lock register into the key resister. Only when the value contained in the key register is equal to the contents of the lock register will the

interface respond to any read or write request directed to it by a software process. Since the user program will not be aware of the combination, only the system software and not the application program will be allowed to access the interface area.

Information is sent via C-bus in the form of messages or packets. Once a computer has placed its message in the output buffer of its interface the C-bus controller will control the transfer of the message to the input buffer of its destination. The bus controller transfers the complete message as a single block and ensures its safe arrival. In this manner, the sending computer is free to perform other duties once its message has been deposited into its output buffer. Secondly, the C-bus controller can monitor message flow and collect statistics for the purposes of decomposition evaluation. This can be accomplished by additional monitoring logic incorporated both on each interface and in the C-bus controller.

CUENET is a reconfigurable network of loosely coupled multicomputers which uses C-bus as its ICN. Fig. 3.4 gives an example where two C-bus structures are used. Each bus of the ICN will have its own bus controller. In addition, every processor will be connected to each bus through a separate C-bus interface. In the case where the ICN consists of two or more buses, each processor using the ICN will have more than one C-bus interface, any of which could



I: C-bus Interface

NMU: Network Memory Unit

Figure 3.4

CUENET: A Reconfigurable Multi-processor based on C-bus

be used to send a message. It is left to the software of each computer to attempt to load all of the buses of the ICN equally. In this manner, the controlling of the use of a multiple bus ICN will be distributed among all the computers.

There are three types of functional units attatched to C-bus that comprise CUENET a master computer, several slave computers, and network memory units (NMU). The master computer is responsible for the coordination of all other computers in CUENET and also acts as the interface between the multicomputer system and the end users. In effect, the master computer contains the major portion of the multicomputer operating system. The computational tasks required by the end users are carried out by the slave processors. The slaves will accept and perform commands issued by the master such as load a user program, obtain a user algorithm from some other location in the network, and start or terminate a user routine. The network memory units are accessible to all the computers of CUENET as a common memory bank.

The slave processors of CUENET need not be homogeneous. In fact, a slave processor can be a special purpose processor such as an associative processor [Kogge 80] or a processor designed for signal processing [Intel 80]. Each slave computer in CUENET will be assigned a PPN or Physical Processor Number. A user task may require an interprocessor

configuration such as the one shown in Fig. 3.5. The user indicates the interprocessor communication structure that he requires as a part of his algorithm. In addition, the user must indicate the characteristics of each processor that will be used to execute his algorithm. These characteristics will be primarily concerned with the physical properties of the slave processors such as the local storage requirements, or the possession of a special purpose computing facility like an arithmetic or fast fourier transform processor. For the purposes of expressing an algorithm the user will assign an LPN or Logical Processor Number for each processor he specifies. Mappings between the LPNs and PPNs are carried out by the resource management subsystem of the operating system which will try to match the processor capabilities requested by the user with the computers that are available within CUENET.

The master processor does not contain any special purpose hardware and thus any computer in our computer unit can function as the master as long as its internal configuration is capable of running the master processor software. The master processor has different rights than the slave procesors. The loading of the user's programs into slave processors and setting up of the access vectors are considered to be some of the responsibilities of the master processor. By allowing the operating system (OS) to perform the mapping from LPN to PPN the flexibility of the operations of CUENET is increased. In addition to those

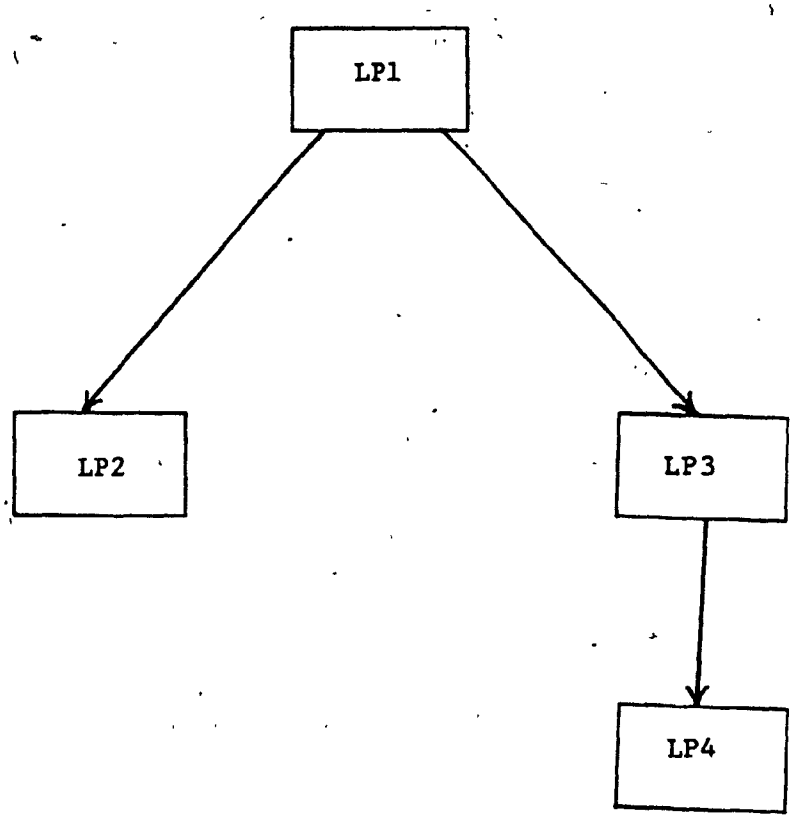


Fig. 3.5 A processor communication structure in the form of a tree.

tasks performed by the OS of a uniprocessor system, the operating system of the master processor is responsible for the following operations:

(a) Aiding the user in decomposing a sequential algorithm,

(b) Allocation of free slaves to user tasks,

(c) Loading of a user process into the slave's memory,

(d) Initialization of slaves and their access vectors,

(e) Handling exceptional conditions such as an attempt by a user task to access a slave for which no access privilidge has been granted.

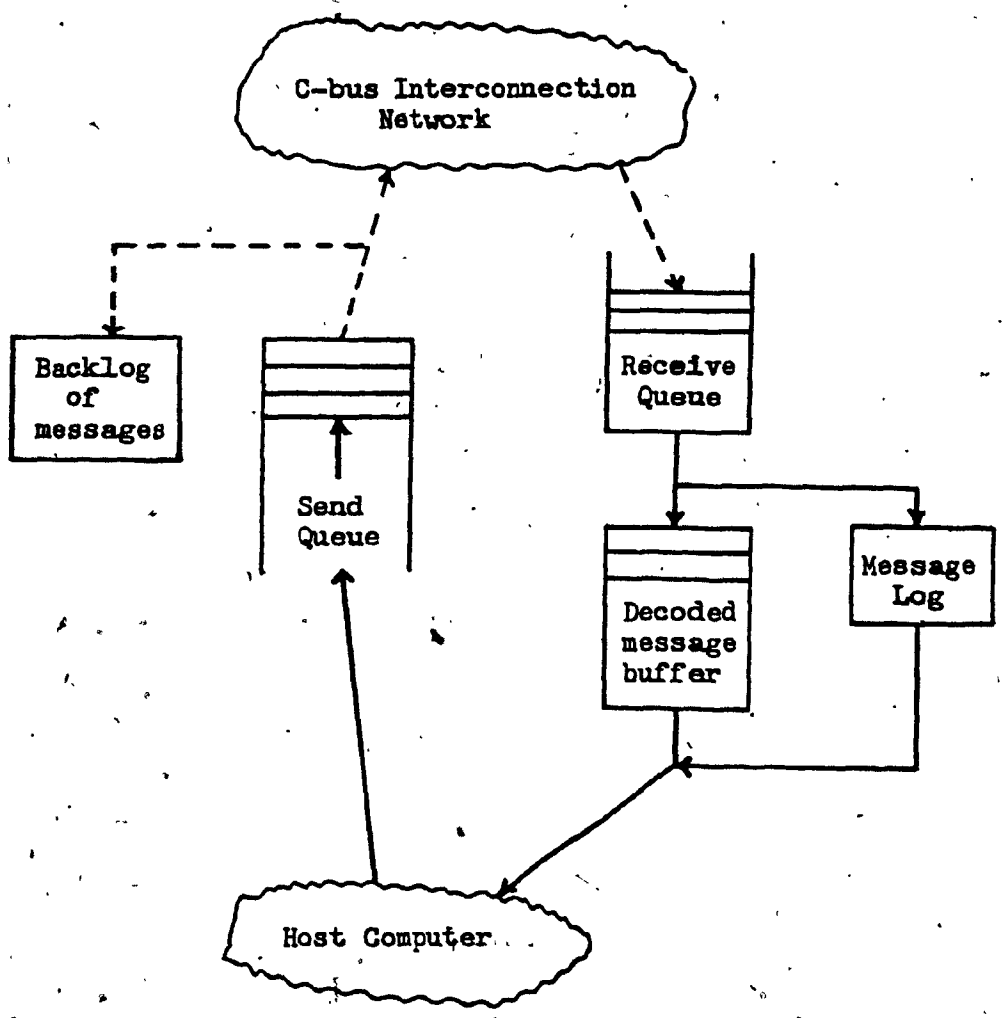
An operating system of this nature will be more complex than an operating system of a uniprocessor system. Because of the clear division of responsibilities between master, slaves, and ICN, we believe the operating system will not be as complex as in the cases of unconstrained multiprocessor systems [Wolf 74].

In many multiprocessing applications, there is a need for several processors to access different parts of a common data bank [Jones 80]. In order to centralize the storage of such "shared data", network memory units are provided in CUENET. Suppose there is a large amount of data common to many processors but only small segments of it are accessed by a processor at a time. Such common data can be centrally stored and maintained in an NMU. Centralized storage and maintenance of common data saves memory because we avoid storage duplication and this will also save time because we reduce the number of update messages transmitted over the

ICN. For the purposes of encoding the transmitted messages and decoding the received messages, each NMU will contain a microprocessor. By associating a processor with each NMU, we can provide a uniform message transfer protocol as seen by the C-bus controllers among the processors and NMU's connected to the ICN. Because these NMU's have an intelligent controller, they can also be used to provide other functions such as synchronization control, searching local data for specific elements and sending the results to other processes, and arbitration between simultaneous read and write requests for the shared data.

3.3 Message Communication

The C-bus interface supports an interrupt driven message system. An interrupt request is issued when the output buffer is empty, or when the input buffer has been filled by the C-bus controller. This will allow the front end communications software for each computer of CUENET to use a message control system as shown in Fig. 3.6. The software modules to create this control system consist of an interrupt handler, a message transmit, and a message receive module. The interrupt handler is responsible for moving messages from the send queue to the output buffer of that computer's C-bus interface and moving messages from the input buffer to the receive queue. The message transmit and receive procedures will be responsible for maintaining the



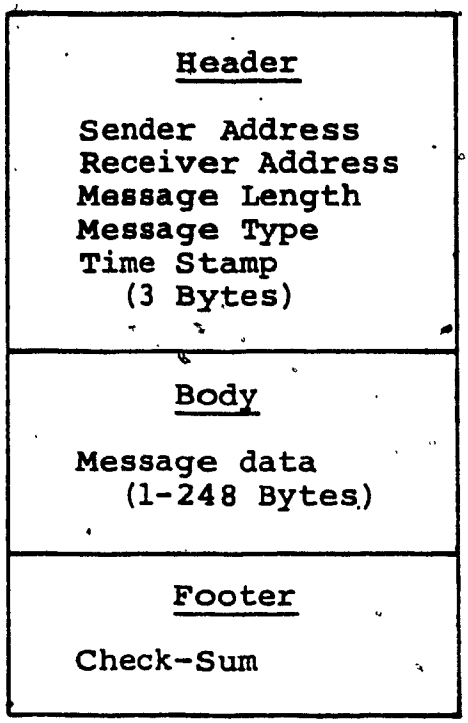
- - - - - : upon interrupt request

Figure 3.6

send queue, decoded message buffer, and the message log. These procedures are given in detail later in this section.

Information is exchanged between computers through one of the buses of the ICN, following a fixed message format. The format as shown in Fig. 3.7 consists of three sections: message header, body, and footer. The time stamp in the message header indicates the time at which the message was mailed by the C-bus controller. The ordered pair < sender address, time stamp > will uniquely identify each message passed in CUENET. The addresses specified in a message will correspond to physical processor numbers, or PPNs, rather than logical processor numbers. The message body may vary in length, but the overall message length should fit into the output and input buffers of the sender's and receiver's interface respectively. The message footer will be used as a checksum.

The verification of transmitted messages is performed in a two step process involving the C-bus controller and the message receive software. During transmission, messages are automatically checked by the C-bus controller by means of a hardware parity checker, and will be retransmitted if necessary. If the C-bus controller, after several tries, decides it cannot send the message, it will notify the master computer of CUENET about the error condition. The second verification step is performed during message decoding when the checksum is processed. If a checksum



Message Type Byte

I	A	E		*	*	*	*

I Intercomputer Message
A Access Vector Load
E Error Message
**** Message Code

Figure 3.7

Message Format

error is trapped by the message receive software, a special message is sent to the originator of the message requesting for a retransmission. As noted in Fig. 3.6, each computer maintains a backlog of the last 'k' messages it has transmitted. If the message requested for retransmission is found in the backlog it is sent a second time, otherwise the master computer is notified that an error has occurred.

When a user task requests a message transfer, it must supply the LPN of the receiver, the starting address of the location the message, and the message length. The message transmit routines will then be invoked to perform the following operations:

S1 Map the LPN of the receiver address to its PPN by using the access vector. If access to the receiver requested by the sending processor is not permitted, create an exception condition and report this to the master computer.

S2 If the message to be transmitted is longer than the maximum message length, divide it into slices. Prepare each slice according to the format specifications and place them into the send queue. Message slices will be numbered as 1 of 3, 2 of 3, etc.

The message receive procedure is invoked by the scheduler when the receive queue is not empty. The steps followed by this procedure are as follows:

R1 Examine the message header to determine if the message is a retransmission request or information for some other software process. If the message requests retransmission of an earlier message, then the backlog of messages is consulted to see if the required message can be found. If the message is found then it is placed into the send queue to await transmission, else an error message is sent to the

master computer.

R2 If the message is not a retransmission request then the check sum is verified. If an error is detected, a retransmission request message is placed into the send queue, else the message header and footer are removed and the message body is placed into the decoded message buffer.

R3 Place an entry in the message log into which a software process will consult when it is waiting for a message. The entry will be constructed from the information found in the message header. The PPN contained in the message header will also be mapped to an LPN using the access vector.

The interrupt handler is activated when either the input or output buffers of the C-bus interface needs servicing. This process is done under interrupt control for the following reasons: The input and output buffers of the interfaces are critical resources in determining the delay of message transmission because they are shared by the C-bus controller and the host computer. No message can be delivered to a computer if its input buffer is not free. In an interrupt driven system, a received message in the input buffer is moved to the receive queue as soon as possible. The steps followed by the interrupt handler are:

I1 Determine if the interrupt is due to an empty output buffer or a full input buffer.

I2 If the interrupt is due to a full input buffer the message is copied into the receive queue. The input buffer is marked as empty which will signal to the C-bus controller that another message transfer to this computer can be performed.

I3 If the interrupt is due to an empty output buffer the next message in the send queue is placed into the output buffer and backlog of messages. The status register is then set to indicate to the C-bus controller that a message is waiting for

transmission.

The C-bus controller will be responsible for the following operations:

B1 When the controller is free, it signals the interface units through the "bus-grant" line. All the interfaces on a bus are daisy chained with respect to the bus grant line. Thus the bus-grant signal will be propagated from one interface to another. With each interface unit a mask bit is provided that can be set or reset by the bus controller. If the mask bit of an interface unit is set, that unit will not respond to the bus-grant signal. This is useful to selectively mask certain interfaces to prevent them from "hogging" the bus.

B2 Assume that a slave has placed a message in its output buffer. When the interface unit attached to this slave receives a bus-grant signal, he does not permit its further propagation. Then he puts his address on the interprocessor data bus to identify himself to the bus controller.

B3 The bus controller determines the destination or receiver address from the header of the message found in the output buffer of the sender.

B4 If the input buffer of the intended receiver processor is free, the message is transferred. Otherwise the bus controller will set the mask bit on the sender's interface unit and enable the bus grant line to propagate further so that another message may be handled.

B5 After transmitting a message, the controller checks the validity of the transmission via the parity bits supplied by the interface units. If an error is found, retransmission is requested before relinquishing the bus from that particular interface unit. After a certain number of trials, if no reliable transmission is possible, an exception condition is created and the master is notified. The master in turn may notify the slave or take corrective action.

B6 If a transmission is completed successfully, the controller marks the output buffer of the sender "empty" and the input buffer of the receiver "full". This enables the interrupt flag of the interface unit of the receiver to signal the slave processor that a message is waiting in his input buffer.

3.4 C-bus Versus Other Multiprocessors

As described in Chapter 2, many attempts have been made to produce multiprocessors, or multicomputers, by various researchers. Each of these systems has its own limitations and the multicomputers that are similar to CUENET in their functions were examined and their shortcomings outlined in sections 2.3 and 2.4. We feel that C-bus is better suited to the needs of current research in the decomposition of algorithms for parallel processing because of its reconfigurability and reduced cost.

The computers that are connected via C-bus are not directly involved with the transfer of messages. In this manner the speed of C-bus is not limited by the speed of the computers. Because the information transfer is achieved through special hardware buffers which the C-bus controller can access without affecting the computers involved in the message transfer, the delay due to contention between the computer and the C-bus controller for memory access is eliminated. Therefore, C-bus can support a wide variety of heterogeneous microcomputers without being affected by the

speed of each individual computer. We have introduced a hardware mechanism (access vector) in order to allow for the reconfiguration of the communication topology between the microcomputers under user program control. As a result, the user is presented with a flexible interconnection network. This will allow him to test various possible decompositions for a given sequential algorithm. Since the control of C-bus is centralized, our interface unit is less complex than the interface units in other systems. Consequently, the per unit cost of our interface unit is low. Thus, we expect that multiprocessors based on C-bus will be more widely used. In C-bus the safe delivery of a message is automatically verified by hardware parity checkers and automatic retransmission is initiated if required.

The C-bus communication system offers two unique security features which are not found in other systems: (1) The access vector: Apart from permitting reconfigurability, it is also used to verify the access privileges of each microcomputer. In this manner C-bus will disallow a user program from sending a message to a computer for which it has no access rights. This allows the C-bus operating system to protect one user program from another. (2) Lock and key registers: They protect C-bus from sending illegal messages generated by unauthorized software.

CHAPTER IV

A SIMULATION OF CUENET CHARACTERISTICS

4.1 Simulation Objectives

Before an implementation of C-bus could be considered, several alternatives and their impact on the operational characteristics of C-bus and CUENET must be examined. In addition, the types and volume of messages that are expected should be taken into consideration. The results of such an analysis would allow us to estimate the capacity of CUENET and determine if our prototype will be capable of supporting the applications for which it is intended. The limitations of C-bus are also interesting because they will be responsible for a large part of the communications overhead encountered by a parallel algorithm executing on CUENET. A software analyst will wish to minimize this overhead and therefore must have some estimates for the communication overhead that can be expected under a given set of conditions.

Our immediate concern was to determine the volume of messages that could be processed by a specific version of C-bus. Factors that we chose to examine under simulation were the size of the input and output buffers, the speed of the memory used for the input and output buffers, the technology used for the bus controller, and the complexity of the special purpose hardware used to construct the bus controller. All these factors will directly affect the

message capacity of C-bus. The sizes and speed of the input and output buffers will determine the largest possible message size and directly affect the length of the message queues at the sender and receiver.

We will consider two possible implementations of the C-bus controller, one using a bit slice microprocessor and the other using a general purpose microprocessor based alternative. The details of each of these two alternatives are given in Chapter 5. As expected the cycle time of the bit slice microprocessor is smaller than that of the general purpose microprocessor and will provide a faster C-bus controller. Since the memory used on the input and output buffers is one of the cost determining factors, we will consider the use of 200 and 500 nanosecond access time memory along with the bit slice based bus controller. With the general purpose microprocessor, the speed of the memory is not an issue because the microprocessor will be the speed determining factor. In this case, we are left with the choice of what form should the special purpose hardare take. Table 4.1 summarizes the various alternatives we considered for the bit slice and general purpose microprocessor based bus controller. Special purpose hardware (SPH II) is a hardware logic circuit which enables the microprocessor based bus controller to transfer one byte of data on C-bus in one instruction cycle. Special purpose hardware (SPH I) is a less complex version of SPH II where the microprocessor based bus controller required two instruction cycles to

transfer one byte of information on C-bus. The values contained in Table 4.1 are estimates derived from analysis of the components used in each particular implementation.

There are two types of messages that can be expected in CUENET. An interprocessor, or type I, message which will normally carry some synchronization information or the output of one user task to be used as input to another. In general, these messages will contain only a few bytes and occur infrequently with respect to the processing time required for a user task. Type II messages occur because of a computer's request for a transfer of a block of information from a network memory unit. These messages usually occur in bursts and generally have a message length equal to the maximum possible message size. The relationship between type I and II messages is shown in Fig. 4.1 where:

S: Average time between two successive bursts of type II messages,

U: Average time between two successive type II messages within the same burst,

T: Duration of a burst of type II message which is a constant,

V: Average time between two successive type I messages.

While the exact values for these variables are not known, we expect that $S > U$ and $U < V$. It is the interaction between

Table 4.1

Estimation of parameter values for different cases
(Time in microseconds)

Alternative	Clock Speed	Transmission time for one byte	Overhead time
A) Micro with special hardware I	1	4	50
B) Micro with special hardware II	1	2	50
C) Bit-slice processor with slow 500 nano second memory	.150	.750	10
D) Bit-slice processor with fast 200 nano second memory	.150	.500	7.5

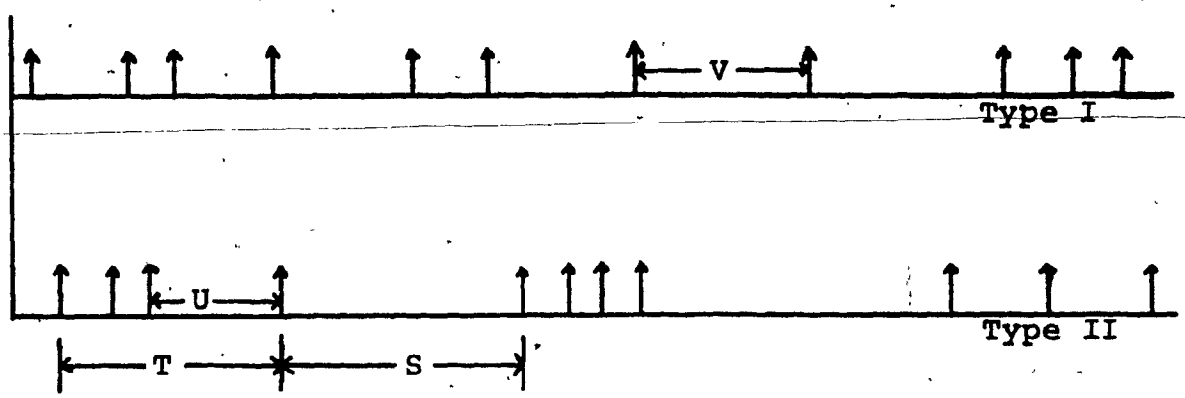


Figure 4.1

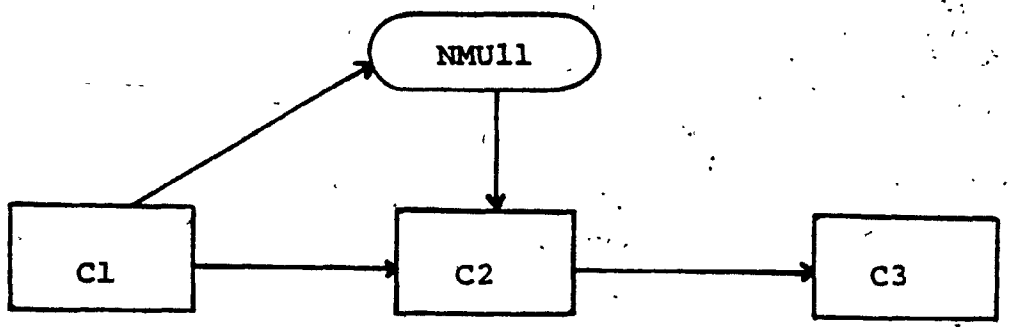
Type I&II Messages

V average time between type I messages
S average time between bursts of type II messages
U average time between two type II messages
T time of a burst of type II messages

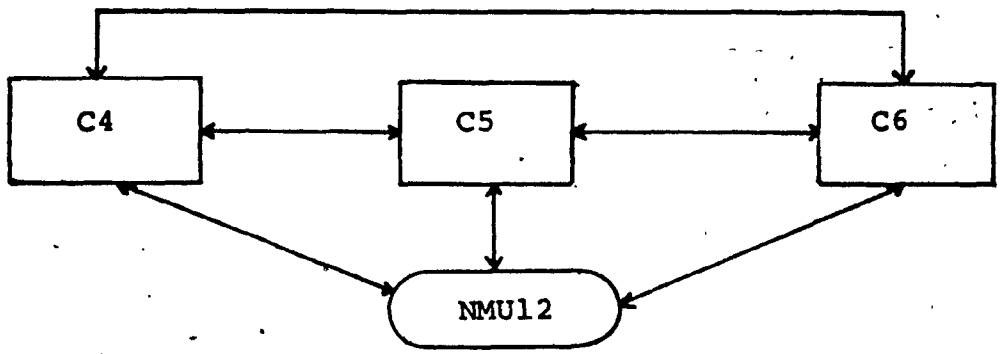
these variables that will determine the volume of message traffic that will be generated by the computers of CUENET.

Finally, the number of computers and types of jobs that will be part of the CUENET prototype must also be considered. Our expectation is that the CUENET prototype would contain approximately 10 computers. Various user algorithms will be running on these computers simultaneously where each user requires certain access patterns between the computers dedicated to his job. Some sample communication topologies are shown in Fig. 4.2(a),4.2(b), and 4.2(c). Algorithms that would require such pipeline and multiple instruction multiple data stream (MIMD) architectures are considered in [Klein 75].

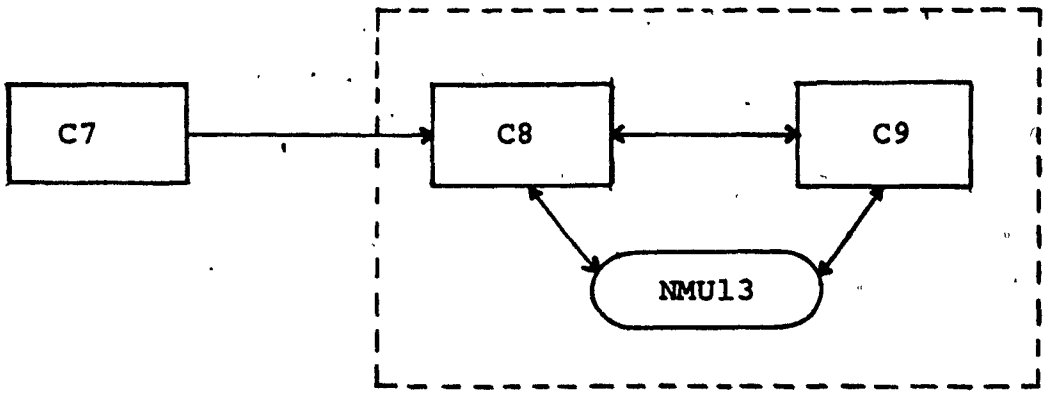
We are interested in determining how the various alternatives for the implementation of C-bus will perform, when we are given a set of user algorithms operating on CUENET under varying message traffic patterns. At this point, we decided that the complexity of the interactions of all these factors would make it impractical to use an analytical method to arrive at estimates for the performance of C-bus. Thus we decided to attempt to develop a queueing model upon which an event based simulation could be programmed. In section 4.2, we explain the model and in section 4.3 the translation of this model into a GPSS program is discussed. Section 4.4 will consider the performance of C-bus as predicted by the simulation



(a) Pipeline Structure



(b) MIMD Structure



(c) Combined Architecture

Figure 4.2

Three user architectures possible on CUENET

measurements.

4.2 Simulation Model

In order to study the communication overhead, we will attempt to model the message flow of CUENET on C-bus at a certain level of abstraction. This model will attempt to describe the dynamic behaviour of the messages expected in CUENET. However, the model will be primarily concerned with how efficiently C-bus processes each message and will not attempt to account for the program conditions under which messages might be generated by any of the computers of CUENET. As a result, messages will only be considered from the time they arrive at an input buffer of a C-bus interface until they are removed from an output buffer.

From the outset of the design of CUENET, we have made the assumption that the time required to load a new user algorithm and reconfigure CUENET to the required architecture is negligible when compared to the computation time of that algorithm. Therefore, we will not incorporate any mechanism into the simulation model to account for the termination or initialization of a user algorithm. We will assume that all user jobs that are active at the start of a simulation run will remain active until the simulation measurements are complete.

We will attempt to model C-bus as a set of queues. The transfer protocol of C-bus is message oriented and therefore

the basic transaction unit that will flow from one queue to another will represent one message on C-bus. Each transaction will have a set of parameters associated with it which will indicate the characteristics of each individual message. This parameter set will contain values to represent the message length, destination address, and sender's address. The message length will be calculated as the sum of a constant, which represents the message header and footer, and a random variable which follows a normal distribution which represents the average type I message length. Type II messages will be fixed at the size chosen for the C-bus interface buffers. The message destination will depend upon the sender and the current CUENET architecture. Under this scheme a message generated by a sender will be sent to any of its possible receivers with equal probability. During the simulation, this choice will be made using a random variable that follows a uniform distribution.

The flow of transactions through the network of queues modeling C-bus can be found in Fig. 4.3. There are three major queueing points in the flow of a transaction from one processor to another: (a) the output buffer of the sender which represents the time a message must wait for the service of the C-bus controller, (b) the C-bus controller which represents the time required to perform the message transfer, and (c) the input buffer to measure the time required by the receiver to empty its input buffer. We will

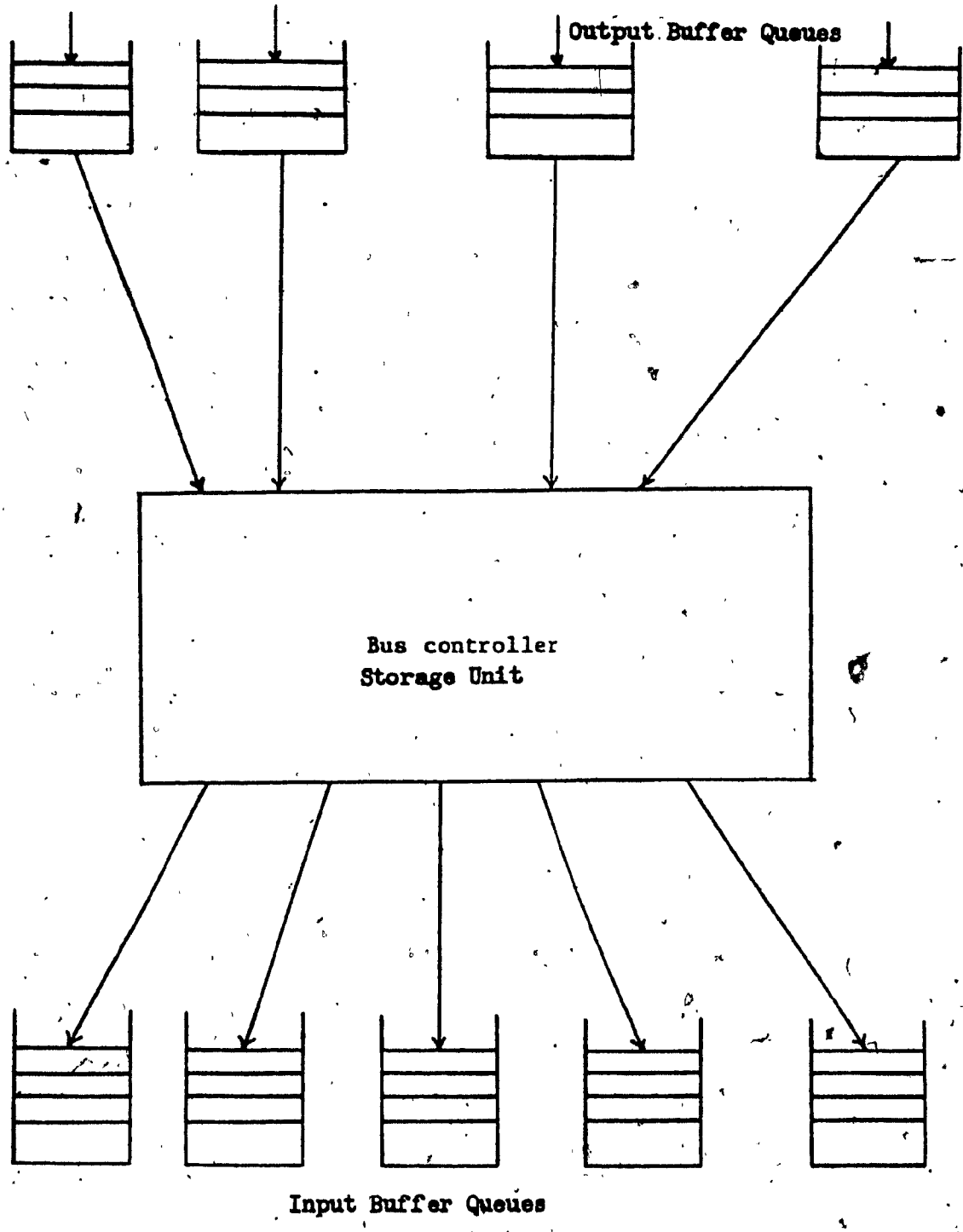


Figure 4.3 A Model for Simulation

monitor these three queueing points to determine how well C-bus is able to handle the message traffic given a specific set of parameters. Our major interest will be in the C-bus controller queue as we expect that this is the point that will be most sensitive to changes in the volume of message traffic. When transactions enter the system they are placed into one of the output queues. When the C-bus controller is free, a single message will move from one of the output queues into the C-bus controller queue where each output queue has equal priority. After a certain delay time calculated from the characteristics of the C-bus controller and current transaction, the transaction will proceed to the input queue indicated by its destination parameter. Once a transaction in the input queue has been processed, it exits the system.

The input parameters to the simulation model will allow us ~~to vary~~ the message traffic and the capacity of the C-bus controller. In this manner we can study the operational characteristics of the various C-bus alternatives under different message volumes. The message volume will be controlled by holding the time of a burst of type II messages (T), while changing the mean time between type II messages (U) constant, and the mix between type I and type II messages. This variation in mix is controlled by varying the mean time between a type I message (V) and the time between a burst of type II messages (S). The random variables S, U, and V are all assumed to follow an

are two parameters that are used to control the speed of the C-bus controller: (a) the amount of time spent performing overhead operations for each message, and (b) the time required to transfer each byte of a message. These parameters have been estimated for the four different alternatives presented in Table 4.1 by analysing the speed of the hardware and the software drivers for the bus controller.

The output from the simulation model will be a set of statistics describing the state of the queueing points. The relevant statistics we will examine for the input and output queues are the average number of transactions waiting in the queues, the number of messages that are not delayed in a queue, the wait time for messages that do get delayed in a queue, and the percent utilization of the C-bus controller. From these statistics we can draw conclusions about the capabilities of the various alternatives for a C-bus implementation.

4.3 GPSS Simulation Program

Once a model had been chosen we then had to consider the feasability of programming the simulation from scratch or attempt to use the constructs of a general purpose simulation package such as GPSS [Gordo 75]. Since our prime interest was to obtain the outputs from our model with as little delay as possible, and it was possible to express our

model as a GPSS program, we decided to use a general purpose simulation language such as GPSS. This section will describe how the system of queues described in the previous section can be expressed as a GPSS program.

The GPSS block diagram of the model is shown in Fig. 4.4 and the key variables used in the simulation program are listed in Table 4.2. The C-bus controller is simulated by a STORAGE block that may contain at most one transaction. The time spent by each transaction in the storage block is governed by the ADVANCE block and the variable BTIME. BTIME is the variable that controls the speed of the C-bus controller during the simulation and is used to switch the simulation between the different bus controller alternatives presented in Table 4.1. The values listed in Table 4.1 are used to calculate the value of this variable for each transaction that passes through the STORAGE block during the simulation. Transactions arrive at the STORAGE block from the queues that represent the output buffers of all the computers of CUENET.

Transactions are created by the GENERATE blocks using the functions MGENI and MGENII. These functions are used to control the generation of type I and type II messages according to an exponential distribution. The average time between type I messages (V) is used as the mean of the function MGEN1 and by changing this mean we can control the volume of type I messages. The average time between type II

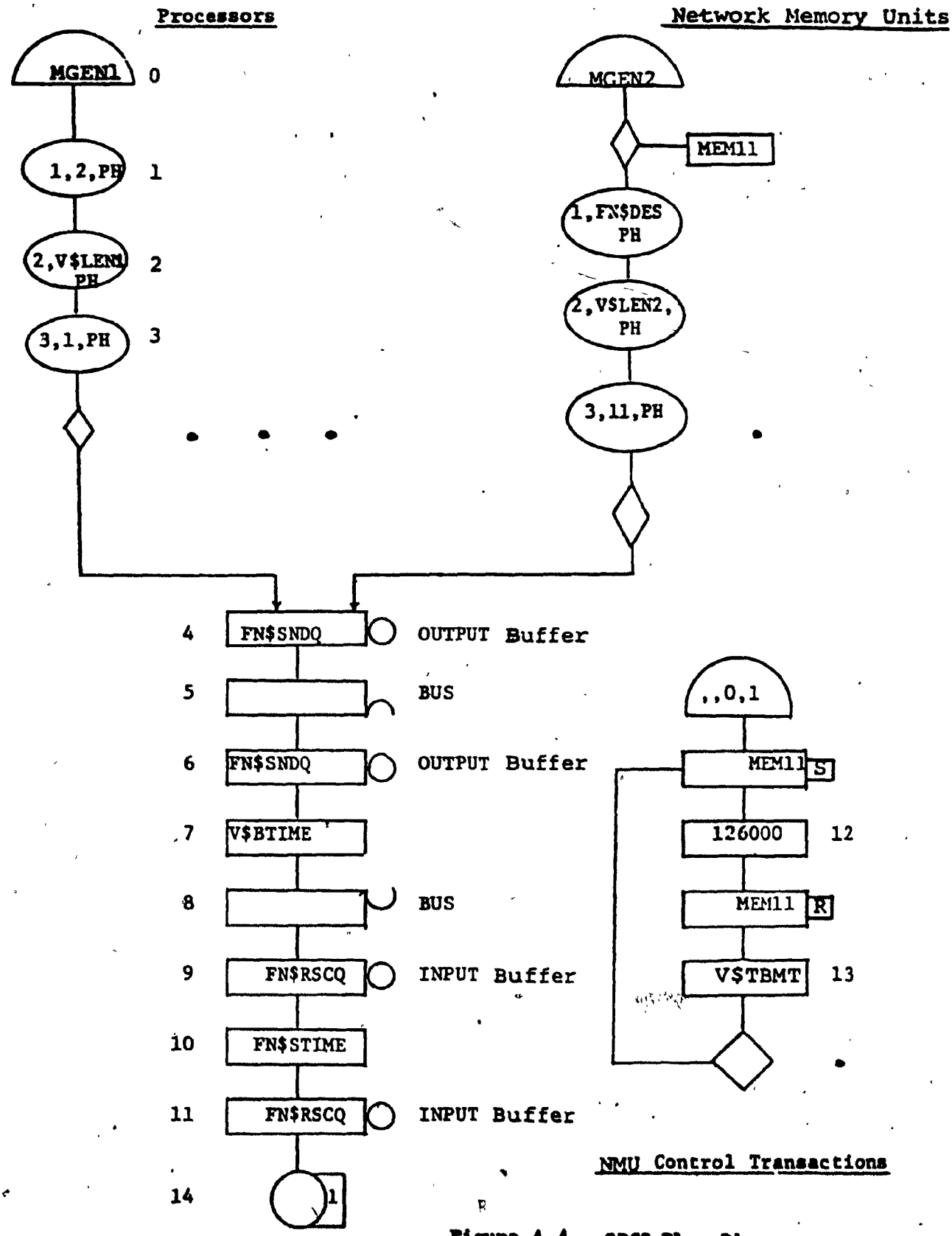


Figure 4.4 GPSS Flow Diagram

Variable	Description
BTIME	Time required by the bus controller to process a message. This depends both on the message length and the message transmission overhead.
STIME	Time required by a receiver to remove a transaction from its input buffer.
TBMT	This variable is used to control the average time between two bursts of typeII messages and follows an exponential distribution.
SNDQ	This function is used to translate the sender number of a transaction into a send queue for the collection of statistics.
RSCQ	Same as SENDQ but for the receive parameter of a transaction.
LEN1	Message length for typeI messages. This function follows a normal distribution with a mean of 15 and a standard deviation of 5.
LEN2	Message length for typeII messages. This function follows a normal distribution with a mean of 64 and a standard deviation of 10.
MGEN1	This function is used to produce transactions that represent typeI messages.
MGEN2	This function is used to produce transactions that represent typeII messages.
DECi	This function will use a random number to choose one out of a number of possible destination queues.

Table 4.2

messages (U) is fixed because we assume that a network memory unit will attempt to send all the messages of a burst as quickly as possible. Therefore, the mean used for this function has been calculated according to the time required by a network memory unit to prepare a message.

The time between a burst of type II messages (S) is controlled by GATE and LOGIC blocks which act as a switch for the GENERATE blocks. The GATE blocks are actually used to start and stop the flow of type II messages under control of the LOGIC block. This is accomplished by creating a control transaction that will continuously circulate in a closed loop, alternately enabling and disabling the flow of type II messages from the GENERATE blocks. This control transaction is generated at the start of the simulation and as it passes through the "S" LOGIC block the GATE is enabled and the "R" LOGIC block will be used to disable the GATE. The ADVANCE block (12), the numbers refer to block numbers in Fig. 4.4, which contains a constant value is used to represent the time of a burst of type II messages (T). We assume that the average size of a block transfer is about 1 K bytes and therefore the value chesen is long enough to allow the required amount of type II messages to be generated. Of course, the number of messages required to transfer 1 K bytes will depend upon the maximum possible message size. The variable TBMT and the second ADVANCE block are used to control the time between a burst of type II messages (5). By adjusting the mean of this

function we will be able to vary the volume of type II messages that are generated.

Each transaction has a parameter block associated with it in which we use three locations to hold the parameter values that define the characteristics of a message. These values are initialized by a set of ASSIGN blocks. The first parameter contains the number of the queue that represents the output buffer of its sender. The second parameter is used to hold the length of the message. This value is obtained from the variable LEN1 for type I messages and LEN2 for type II messages. The means used for these two functions will depend upon the size of the input and output buffers of the C-bus interface units and the average number of bytes expected in the type I messages generated by the computers of CUENET. The third parameter contains the number of the queue that represents the input buffer of its destination. According to Fig. 4.2 we notice that in some cases a sender will be capable of generating messages for more than one destination. In order to realize this in the simulation, a function (Deci) is used to determine the value of this parameter. If a computer of CUENET can send messages to more than one processor, we assume that the probabilities of sending a message to each of the possible destinations are equal.

Upon generation, a transaction will pass through a set of ASSIGN (1,2,3) blocks where the characteristics of each

message are defined. The transactions then enter the queue that represents the output buffer of the sender via a QUEUE block (4) and will wait there until the STORAGE block representing the C-bus controller is free. The transactions will then enter the STORAGE block by passing through an ENTER block (5), depart the output buffer of its sender (6), and wait for a specific amount of time which represents the transfer time of a message on C-bus in an ADVANCE block (7). The transaction will then leave the STORAGE unit via a LEAVE block (8) and enter the queue of the input buffer of its destination (9). The transaction will wait a certain amount of time (10), which represents the time required by the destination computer to empty its input buffer that is controlled by the variable STime. Finally, the transaction is removed from the input queue (11) and passes through the TERMINATE block (14), where it is removed from the simulation.

The GPSS program listing can be found in Appendix I.

4.4 Simulation Results

The simulation was run for all of the four alternatives listed in Table 4.1 while varying the traffic of type II and type II messages as follows:

ALT1&ALT2: We hold the average time between type I messages (V) fixed while the average time between bursts of type II messages (S) is varied. Then the same set of experiments is performed with the average time between type I messages varied (U) and the average time between bursts of type II messages (S) fixed. Both of these two sets of experiments were performed for the case when two user algorithms requiring architectures shown in Fig. 4.1 (a&b), and three user algorithms requiring the architectures shown in Fig. 4.2 (a&b&c), are active in CUENET throughout the simulation.

ALT3&ALT4: We hold the average time between type I messages (V) fixed while the average time between bursts of type II messages (S) is varied. Then the same set of experiments is performed with the average time between type I messages varied (U) and the average time between bursts of type II messages (S) is fixed. For this alternative we considered the case where three user algorithms are active. The two algorithm case is considered to be less important because of the speed of the alternatives being studied.

All the time scales used in the following graphs are in units of milliseconds, unless otherwise specified. When type II messages are being varied, the scale used is in units of T (the average time of a burst of type II messages). The value plotted is S (the average time between a burst of type II messages) which is used to control the mix of type I and type II messages.

Fig. 4.5 and 4.6 are two samples of the information that is obtained from a set of simulation runs for one of the possible C-bus implementations. In both cases, the average time between type I messages is varied while the average time between bursts of type II messages is fixed at

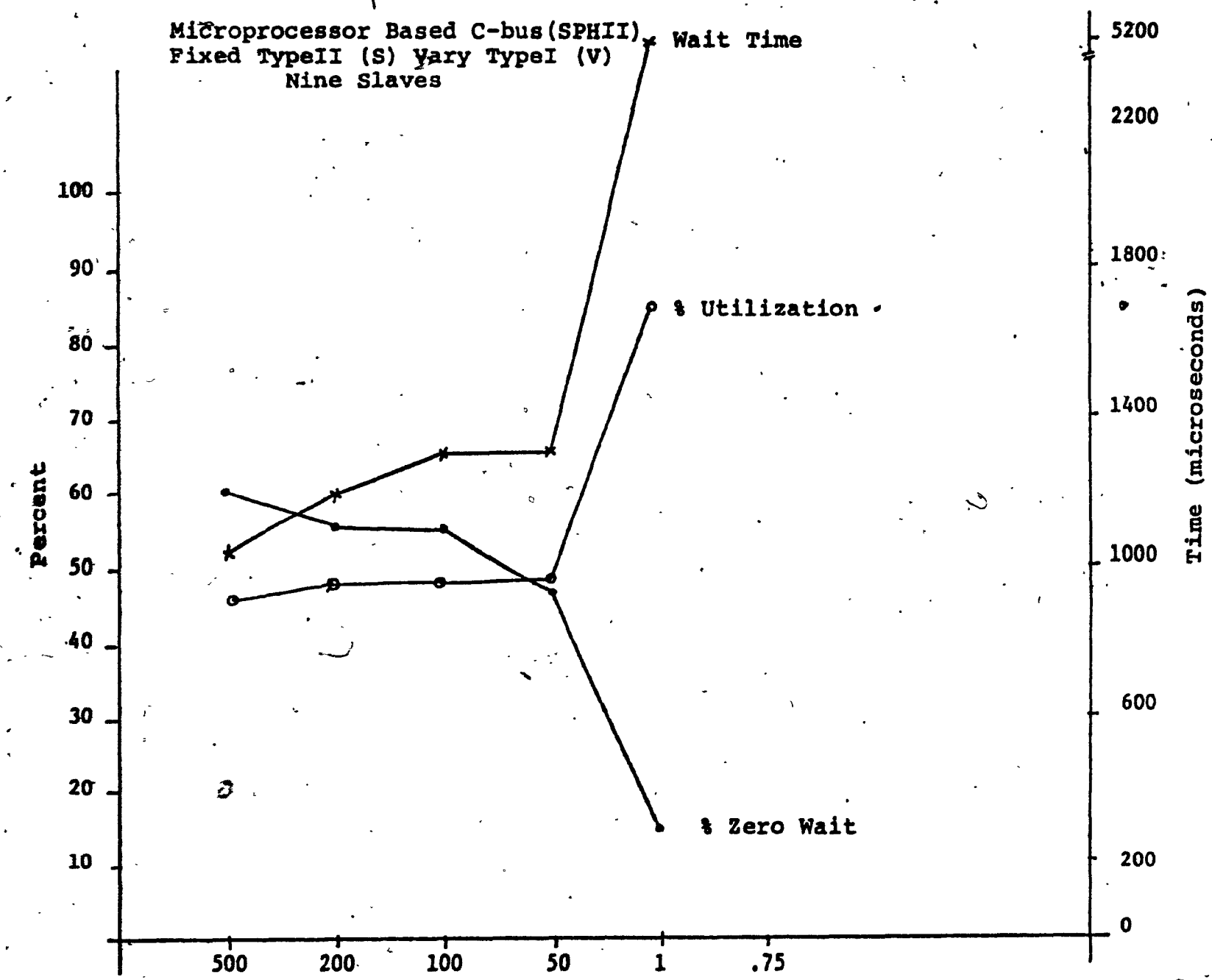

Microprocessor Based C-bus(SPHII)
Fixed TypeII (S) Vary TypeI (V)
Nine Slaves
Wait Time
% Utilization
% Zero Wait
Percent
100
90
80
70
60
50
40
30
20
10
Time (microseconds)
5200
2200
1800
1400
1000
600
200
0
500
200
100
50
1
.75
Ave. Time Between TypeI Mess. (V)

Figure 4.5

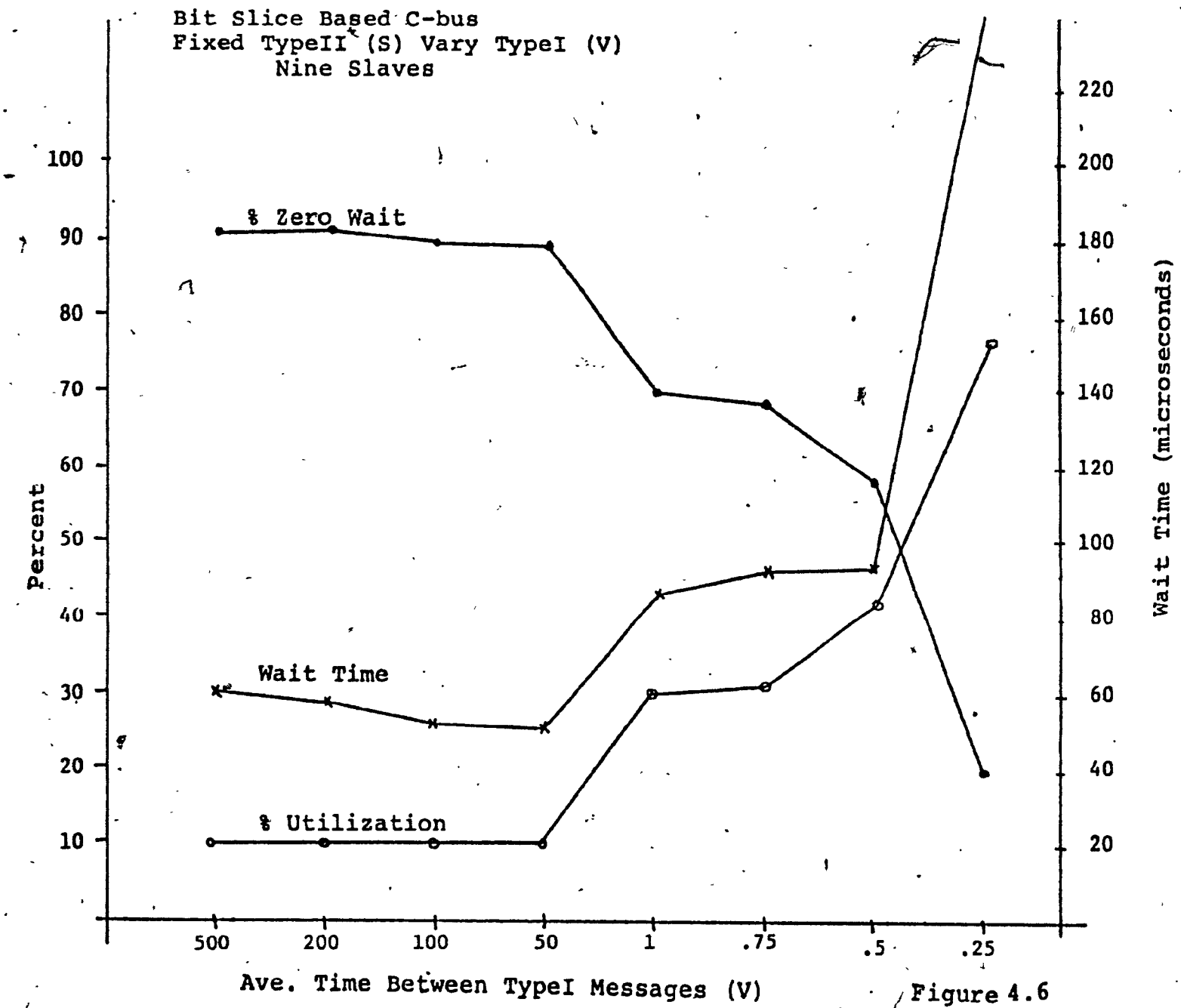

Bit Slice Based C-bus
Fixed TypeII (S) Vary TypeI (V)
Nine Slaves
% Zero Wait
Wait Time
% Utilization
Percent
100
90
80
70
60
50
40
30
20
10
Wait Time (microseconds)
220
200
180
160
140
120
100
80
60
40
20
500
200
100
50
1
.75
.5
.25
Ave. Time Between TypeI Messages (V)

Figure 4.6

0.2T. Each figure contains three graphs, the percentage of transactions that gain immediate access to the C-bus controller, the average wait time for those transactions that are required to wait for access to the C-bus controller, and the percentage utilization of the C-bus controller. As expected, the bit slice based C-bus performed better under the same load conditions as the microprocessor based C-bus. In both cases, the relationship between the three graphs were similar where the waiting time for a transaction and the number of transactions that must wait increases as the utilization of C-bus approaches 100 percent.

In Fig. 4.7, 4.8, 4.9, and 4.10 the results of the other sets of the simulation runs are shown for the various possible C-bus implementations. Only the percent utilization of C-bus is shown, however, the relationship between these graphs and the percent zero wait and average wait time are still the same as those of Fig. 4.5 & 4.6. In all cases, the bit slice based C-bus exibited superior performance however the microprocessor based C-bus is capable of supporting between six and ten computers if their usage of C-bus is not extremely heavy. Both the microprocessor and bit slice based C-bus controllers showed greater sensitivity to a change in the average time between a burst of type II messages (S) than in the average time between type I messages (V).

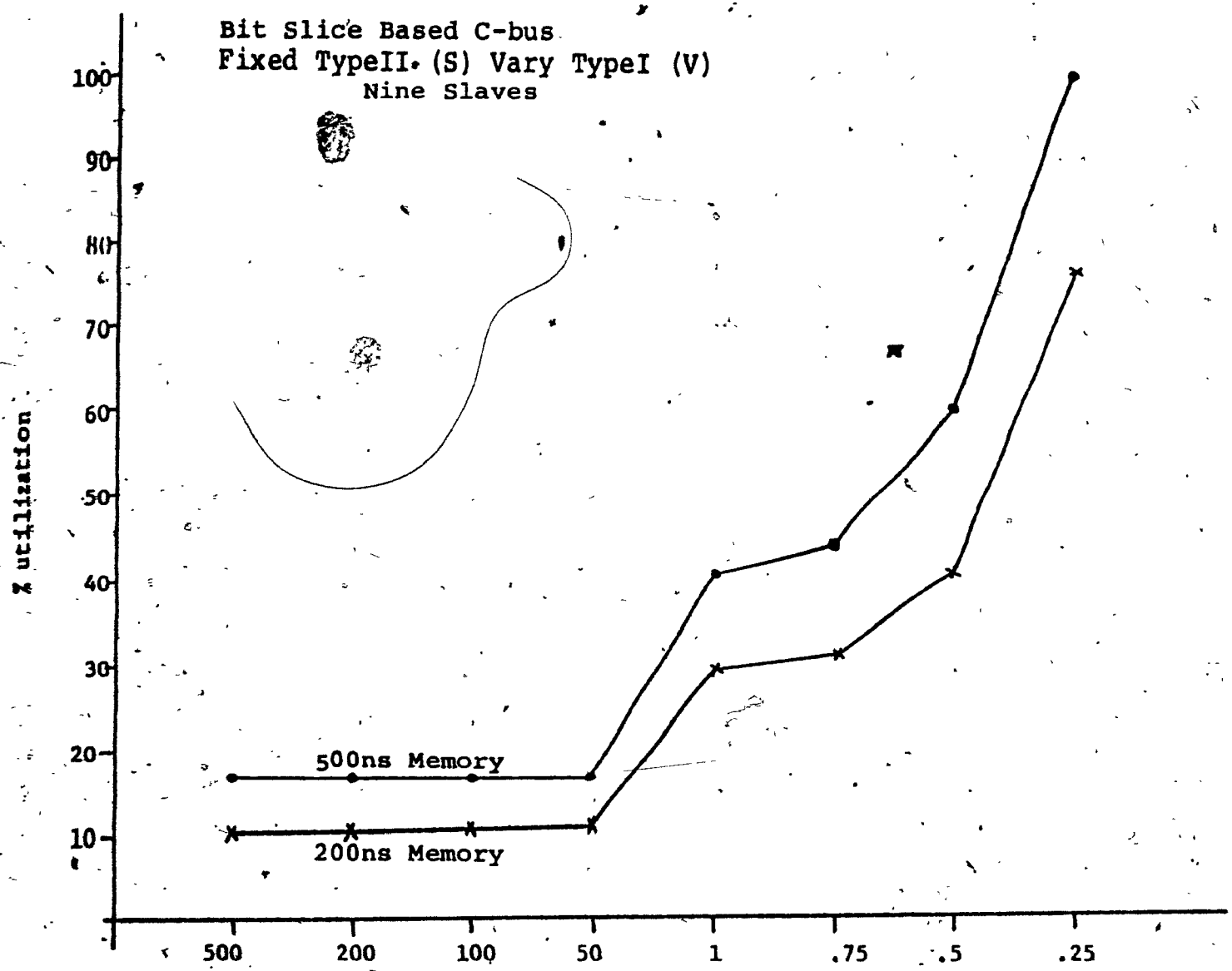

Bit Slice Based C-bus
Fixed TypeII (S) Vary TypeI (V)
Nine Slaves
% utilization
100
90
80
70
60
50
40
30
20
10
500ns Memory
200ns Memory
500
200
100
50
1
.75
.5
.25
Ave. Time Between TypeI Messages (V)

Figure 4.7

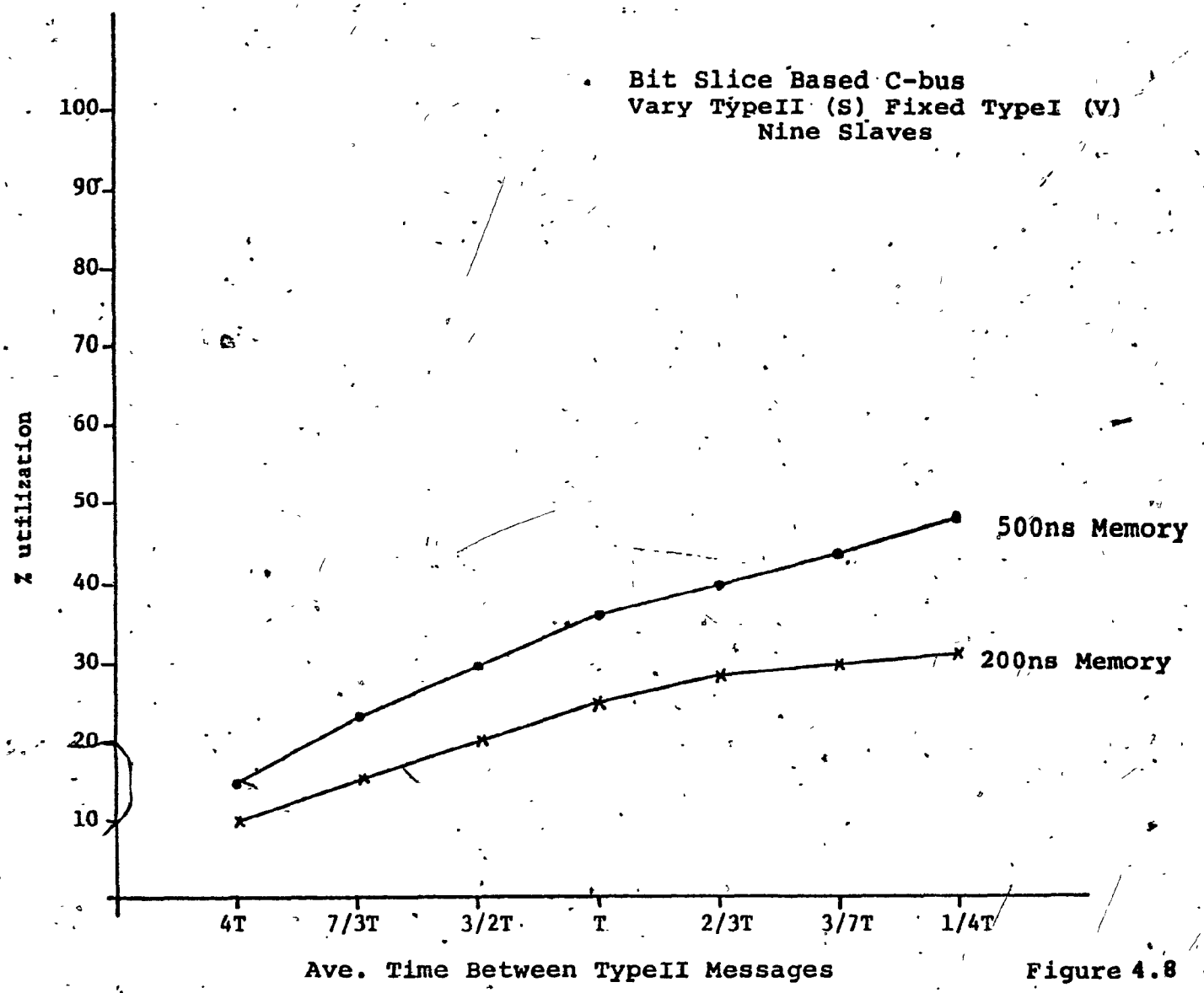

Bit Slice Based C-bus
Vary TypeII (S) Fixed TypeI (V)
Nine Slaves
% utilization
100
90
80
70
60
50
40
30
20
10
500ns Memory
200ns Memory
4T
7/3T
3/2T
T
2/3T
3/7T
1/4T
Ave. Time Between TypeII Messages

Figure 4.8

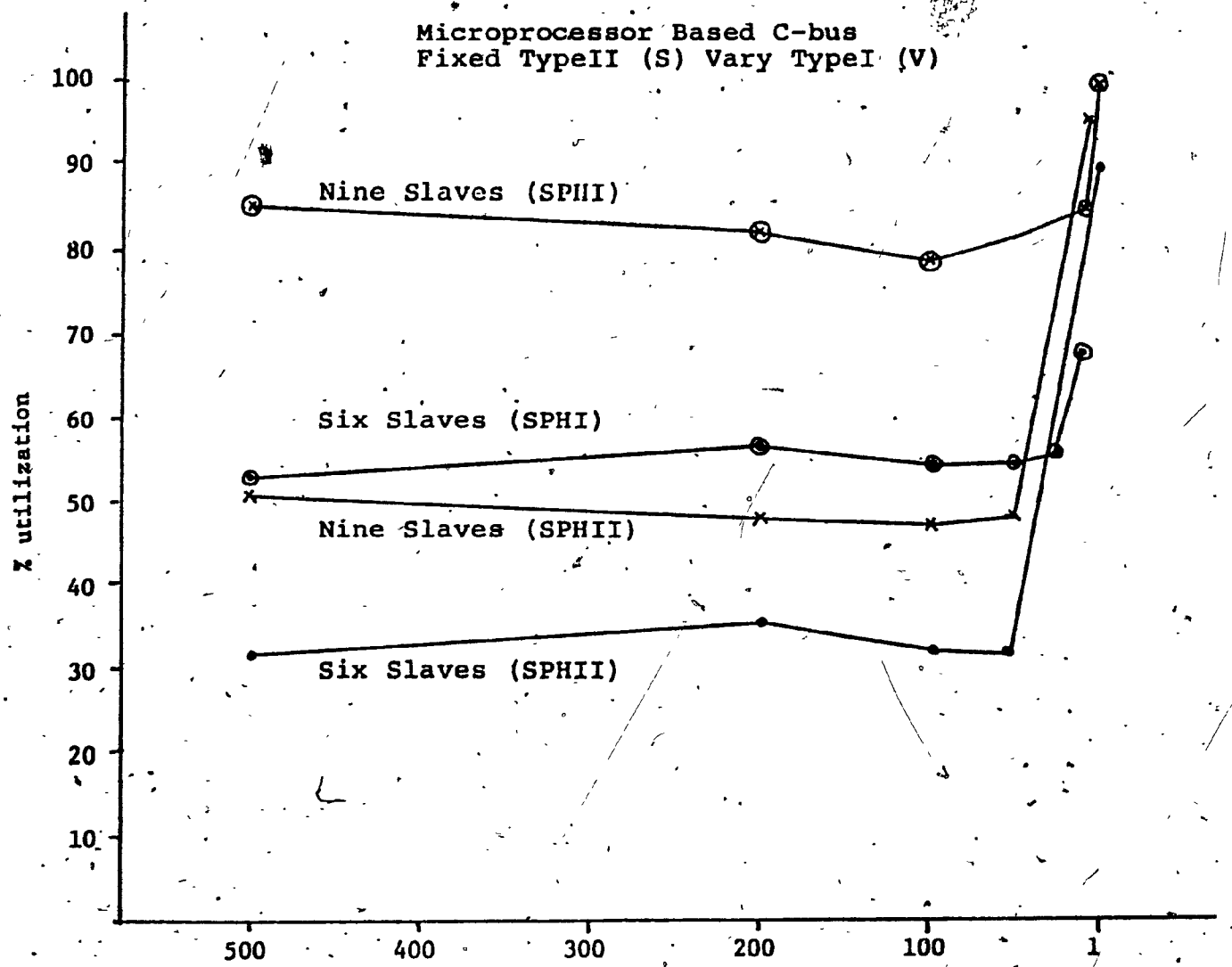

Microprocessor Based C-bus
Fixed TypeII (S) Vary TypeI (V)
Nine Slaves (SPHI)
Six Slaves (SPHI)
Nine Slaves (SPHII)
Six Slaves (SPHII)
% utilization
100
90
80
70
60
50
40
30
20
10
500
400
300
200
100
1
Ave. Time Between TypeI Messages

Figure 4.9

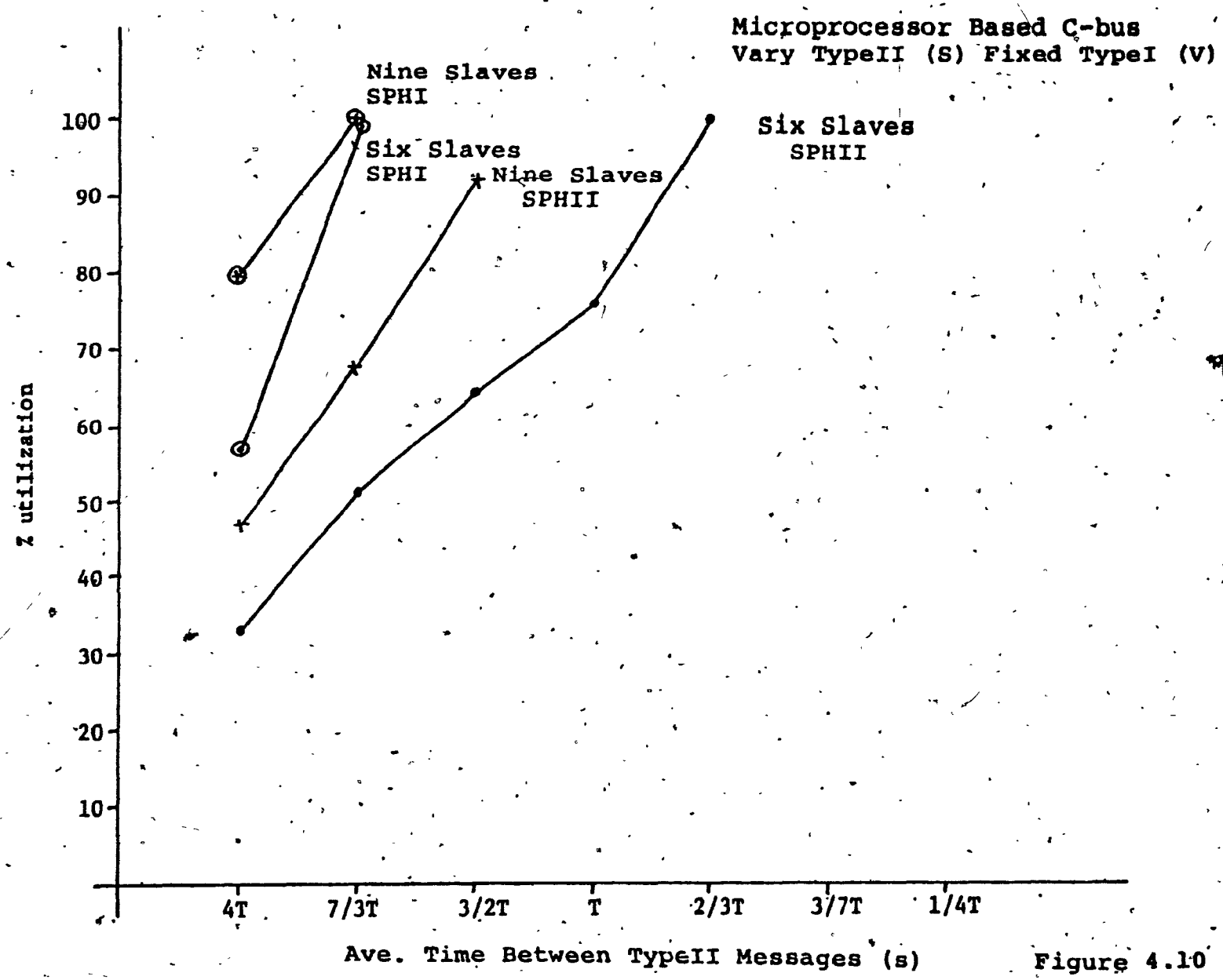

Microprocessor Based C-bus
Vary TypeII (S) Fixed TypeI (V)
Nine Slaves
SPHI
Six Slaves
SPHI
Nine Slaves
SPHII
Six Slaves
SPHII
% utilization
100
90
80
70
60
50
40
30
20
10
4T
7/3T
3/2T
T
2/3T
3/7T
1/4T
Ave. Time Between TypeII Messages (s)

Figure 4.10

Upon examining Fig. 4.7, a curious development seems to occur at an interarrival time of one millisecond. It should be noted that the axis represents the interarrival time rather than the number of messages generated. Upon closer inspection, one can see that the number of messages generated by each slave processor increases very rapidly as the interarrival time approaches one millisecond, as shown in Fig. 4.11. We also see that this is followed by a large drop in the number of messages generated by NMU's. This can be explained because each simulation run is complete when a certain number of transactions have been processed but not at the end of a specific time period.

As the average time between type I messages sent by the same slave processor tends towards one millisecond (Fig. 4.9), the utilization curve reaches a saturation point and the waiting time (not shown) increases enormously. Within this one millisecond time period there will be approximately one interprocessor, or type I, message transfer request from each slave processor, each requiring 100 microseconds service time from the bus controller. When the interprocessor message transfer requests are combined with the requests from two NMUs which require about 175 microseconds of the bus controllers time per request, the total service time demand on the bus controller exceeds the available time. It is well known in queueing theory [Klein 75] that under this condition the waiting time increases indefinitely.

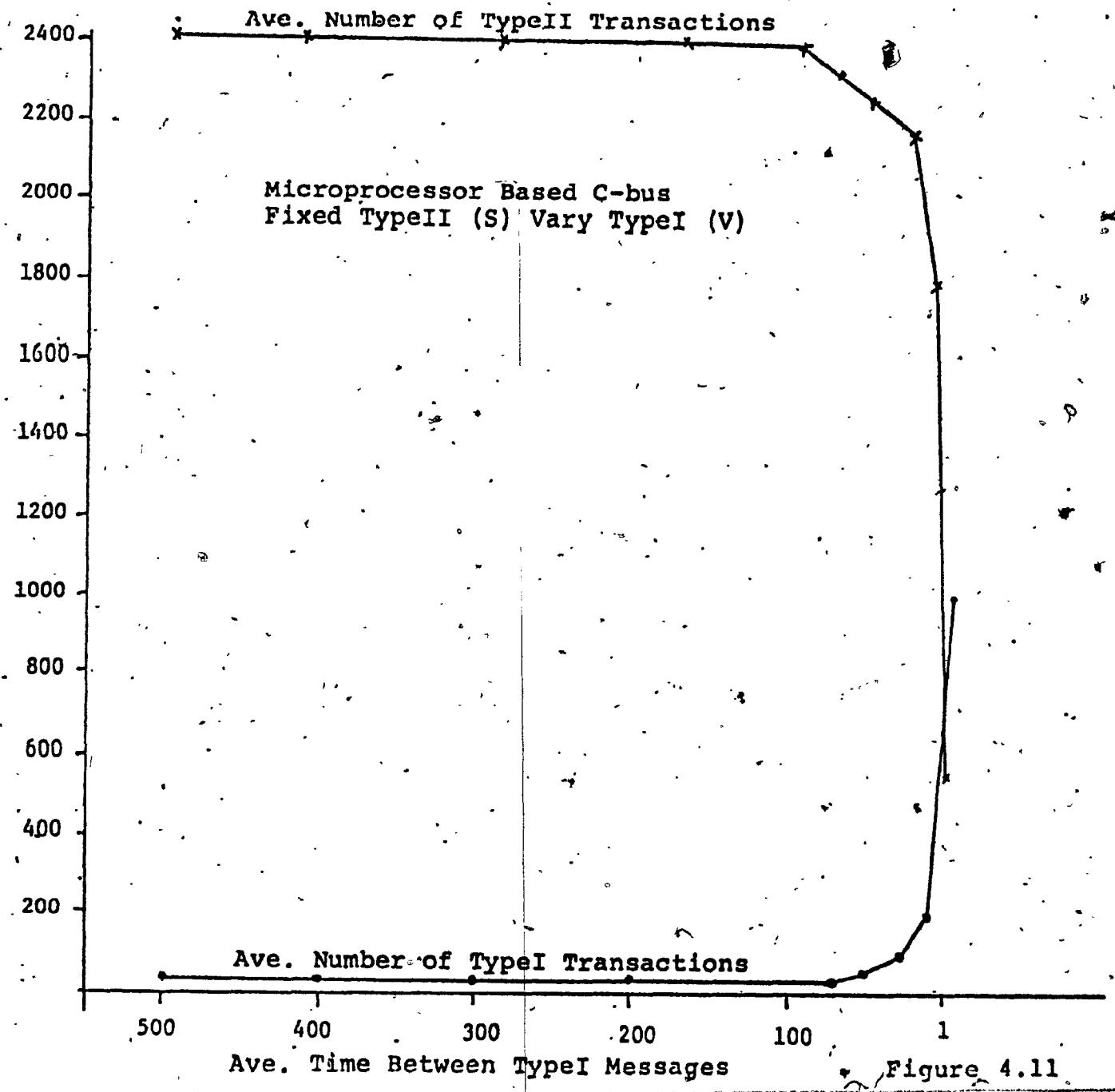

Ave. Number of TypeII Transactions
Microprocessor Based C-bus
Fixed TypeII (S) Vary TypeI (V)
Ave. Number of TypeI Transactions
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
500
400
300
200
100
1
Ave. Time Between TypeI Messages

Figure 4.11

Ave. Time Between TypeI Mess.	Sender Queue	Receive Queue
500 ms	.001	.07
200 ms	.002	.07
100 ms	.002	.08
50 ms	.005	.08
25 ms	.008	.08
11 ms	.02	.08
1 ms	5.1	.12

Average Message Size
TypeI 25 Bytes
TypeII 64 Bytes

Table 4.3

Average Number of Messages in Queues for Six Computers Using a Microprocessor Based Based C-bus (SPHII)

Table 4.3 presents the average number of transactions in the queues representing the input and output buffers during various simulation runs for the microprocessor based C-bus controller. The mean type I message length was set at 25 bytes and the mean length of a type II message is 64 bytes. The queues only became very large when the saturation point of the C-bus controller had been reached, otherwise a buffer size of 64 bytes is shown to be adequate.

One way to validate a simulation model is to compare the simulation results with actual measurements made on a running system. This approach in our case is not possible as the architectures we have simulated are not yet completely built. However, we find the overall trend of the simulation outputs are in agreement with the fact that increasing load on the system results in decreasing response time and increasing utilization of the resources. Also, we find that the saturation point of one millisecond is explainable as it is the point where the average transaction arrival rate exceeds the average service rate. Since we have been able to explain all the trends observed at this point, we assume that our initial model is valid and that the results obtained from the simulation runs are relevant.

CHAPTER V

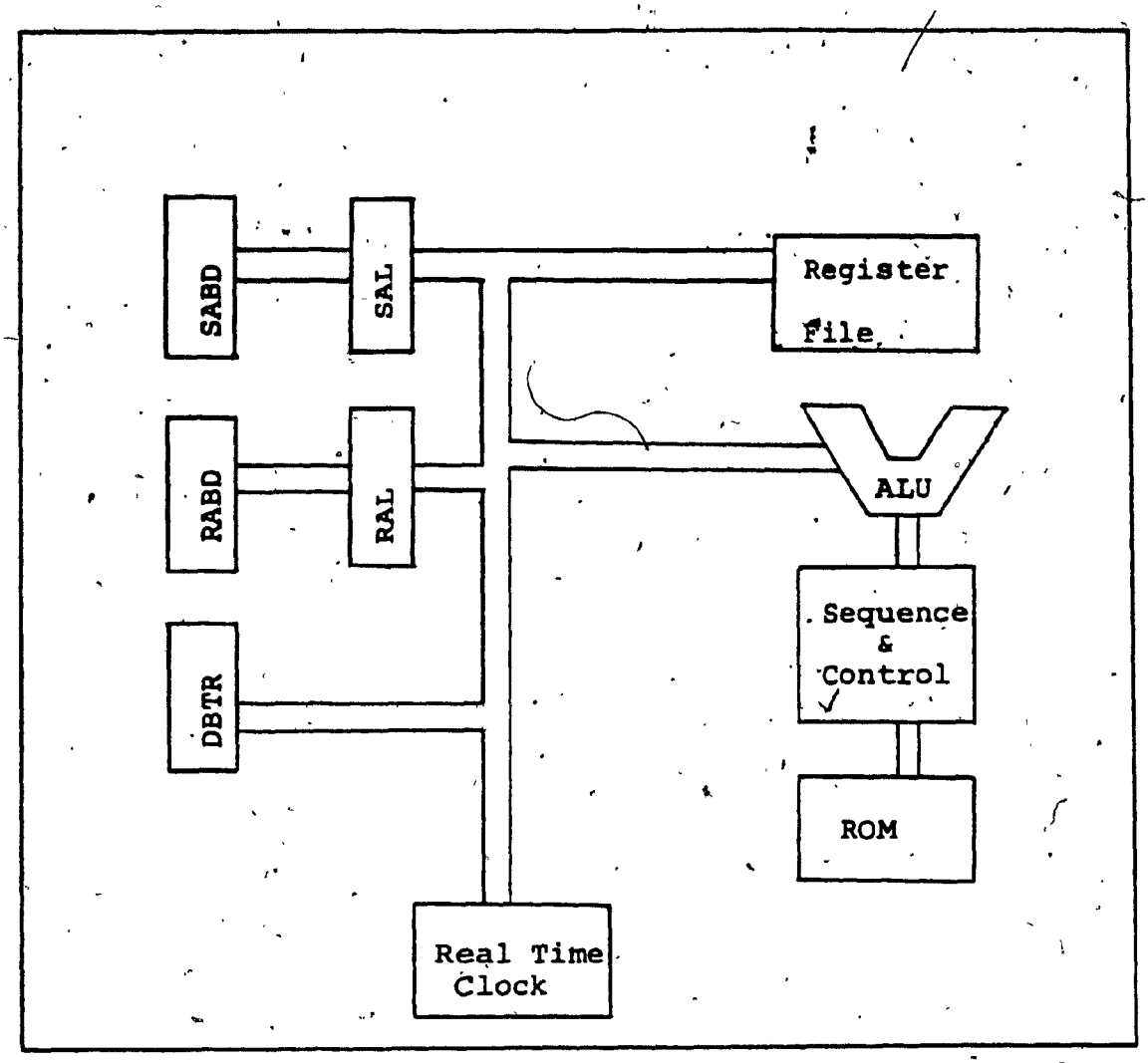
THE DESIGN AND IMPLEMENTATION OF CUENET

5.1 Design Tradeoffs

The design and implementation of a multicomputer system involves several tradeoffs. In this section the various tradeoffs that were analysed during the construction of C-bus and CUENET are outlined. In many cases when examining a specific tradeoff the physical resources at our disposal played a key role in choosing an alternative, in addition to the design goals discussed in section 3.1.

We find that there are at least three ways of implementing the C-bus controller: (a) using bit slice microprocessors, (b) using a general purpose microcomputer, and (c) by means of dedicated special purpose hardware. The latter alternative would give rise to a very fast C-bus but the controller circuitry would be complex and it would be difficult to modify the functions of the controller. The entire circuit would have to be designed, constructed, and tested in house which would require a large amount of manpower and could not be achieved within the time constraints of a master's thesis.

A C-bus controller using bit slice technology could be constructed as shown in Fig. 5.1. The controller will contain most of the standard components of a CPU such as a register file, an arithmetic and logic unit, a microprogram



DBTR-Data Bus Transceiver
RABD-Receive Address Bus Driver
SABD-Send Address Bus Driver
RAL-Receive Address Latch
SAL-Send Address Latch

Figure 5.1
Bit Slice (Microprogrammed)
Based C-bus Controller

sequencer unit, and the microprogram ROM. In addition, the bus controller will require registers and drivers through which it can drive the C-bus address lines, and interact with the C-bus data lines. The bus controller will be able to sense the status of all units on C-bus via the status lines connected to the microprogram sequencer unit and send control signals through the control register.

The C-bus controller could also be implemented using a microcomputer with some special purpose adaptation hardware. In this case we were faced with the problem of what cost-effective functions should be put into the additional hardware or performed by the microprocessor. Of course, as more functions are handled by extra hardware the faster the bus controller will operate but the complexity of the circuitry will also increase. The C-bus controller uses the programmable processor to coordinate the operations of the special purpose hardware modules. However, simple tasks will be done as much as possible using custom circuitry so that a reasonable bus speed can be obtained. An example of this is an auto increment feature added to the C-bus address control module. If the incrementing of the C-bus address latches were to be performed by the programmable processor, several instruction cycles would be required, while the additional hardware can perform this function in less than one instruction cycle. Because the incrementing of the C-bus address latches must be performed for each byte transfered on the C-bus, a major increase in the transfer

rate is achieved by incorporating this function into the special purpose hardware.

As shown by the simulation described in the previous chapter, the bit slice based C-bus controller is faster and capable of supporting a greater transfer rate than the microprocessor based C-bus controller. However, with the microprocessor based controller we could still support a sufficient amount of computers provided that each computer limits its usage of C-bus. Since we did not possess the resources required for a bit slice implementation and the simulation had shown a microprocessor based controller to be adequate for our CUENET prototype, we have chosen to use a general purpose microcomputer and constructed the required hardware extensions for it to function as a bus controller. The modifications involved the addition of such functions as address control, data transfer, and arbitration for C-bus. All the extra hardware circuits fit into a single wire-wrapped board which in turn fits into one slot of the mother board in the bus controller computer. A detailed description of these extensions are found in section 5.2.

The choice of buffer sizes is yet another design issue. In our design, the input and output buffers of an interface unit are a reserved part of the memory address space of the host computer. Suppose the buffer size is selected as large as 1024 bytes. If the messages transmitted are mostly much smaller than the buffer size, then the reserved address

space will not be well utilized. On the other hand, a small buffer size will require too many message packet transfers for a given message length. The simulation indicated that a buffer size of 64 bytes would be adequate and after some considerations, we chose a buffer size of 256 bytes which is also the sector size of the floppy disks in our computers.

Since all the computers connected to a C-bus are daisy chained together, it is possible to have the problem of livelock or starvation [Ullma 80]. We have provided a mask bit in each interface that can be set or reset by the bus controller. The starvation problem is handled by programs in the C-bus controller, described in section 5.4, by suitably setting and resetting the mask bits. In this manner the starvation avoidance mechanism is flexible and can be adjusted as required.

How many address lines should be there in C-bus? If we use one address bus and transfer one byte of data from the sender to receiver, it will require two steps as shown in Fig. 5.2(a). In step 1 one byte is read from the output buffer of the sender to a local buffer of the bus controller, and in step 2 the byte is placed into the input buffer of the receiver. Another approach is as shown in Fig. 5.2(b). In one cycle of the programmable bus controller one byte of data is transferred on C-bus. However, two distinct address buses are required. It is also possible to use one set of address lines in a

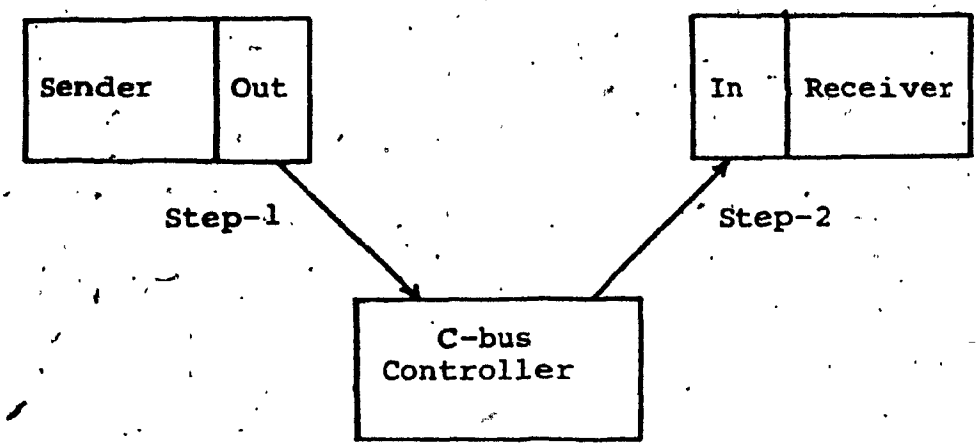


Figure 5.2(a)

Double step transfer

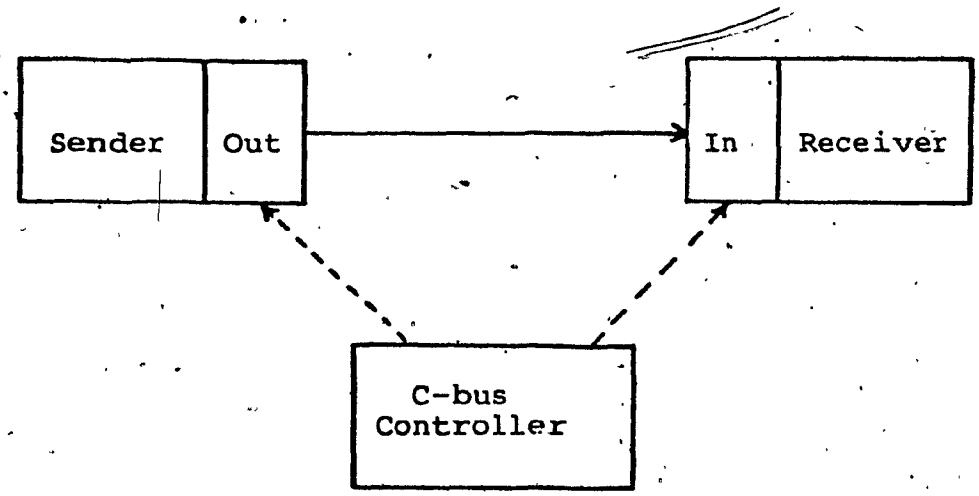


Figure 5.2(b)

Single Step Transfer

multiplexed mode so that they can carry both the send and receive address information. The multiplexing of the address lines will result in a reduction in the total number of lines required which will also reduce the cost per unit length of the C-bus cable. However, the hardware will be more complex and the transmission time for a single byte will be longer than that of the two address bus solution. We decided to use the two address bus option for C-bus. C-bus is not designed to run several kilometers and hence the additional cost due to the increased number of wires in the address lines was not a significant factor in our case.

In C-bus there are two sets of address lines each containing 16 wires. The 16 bits are divided into two segments, a most significant segment (MSS) and least significant segment (LSS). The number 2**MSS determines the maximum number of computers that can be connected to one C-bus and 2**LSS indicates the total size of the "interface memory address space" or IMAS. The size of IMAS must be large enough to accommodate the space required by the input and output buffer, the access vector, and the control and status registers. As per the present design, we can connect a maximum of 128 computers to one C-bus. The number of address lines in the MSS represents a trade off between the cost of extra address lines and the total number of computers that can be attached to a single bus. The choice of the number of address lines of the LSS is governed by the size of all the hardware modules found on the interface

card. If the number of address lines is critical, the size of LSS may be reduced by multiplexing.

The total number of address lines on C-bus is 32. Besides which, there are nine lines used for data transmission and parity. There are eight control lines on C-bus which are used for data transfer control and C-bus arbitration which are described in section 5.3. This gives us a total of 49 signal lines. In order to reduce the signal degradation over the length of C-bus, we have employed a twisted pair (signal and ground) to carry each signal.

To choose a synchronous or asynchronous data transfer protocol is yet another design problem. The former is faster, simpler, and less costly, whereas the latter provides greater reliability. In message communication protocols, the bottom most layer that supports all other layers is the physical layer [Tanen 81], and it should be implemented as efficiently and reliably as possible. To achieve this efficiency in the C-bus design, we use an asynchronous or handshake approach for the selection of the next sender to be serviced. Once the sender and receiver are established, the entire message is transmitted using a synchronous protocol. Thus we attempt to achieve a balance between the reliability of an asynchronous protocol and the speed of a synchronous transmission.

Both C-bus and CUENET employ centralized control

mechanisms. The advantage of a centrally controlled system is a reduction in the software complexity, but a certain amount of reliability is sacrificed. The bus controller computer, in our case, is a general purpose microcomputer whose reliability is quite high. Also, the adaptation, or modification, hardware is carefully designed and well tested. If reliability is extremely important in a particular implementation, as our design permits, we could use multiple C-buses in a CUENET. If the CUENET master fails the down time of the network is made minimal because the master software can be quickly loaded into any other computer and then CUENET can be restarted. This interruption in service is the price paid for the simplicity gained in the software design.

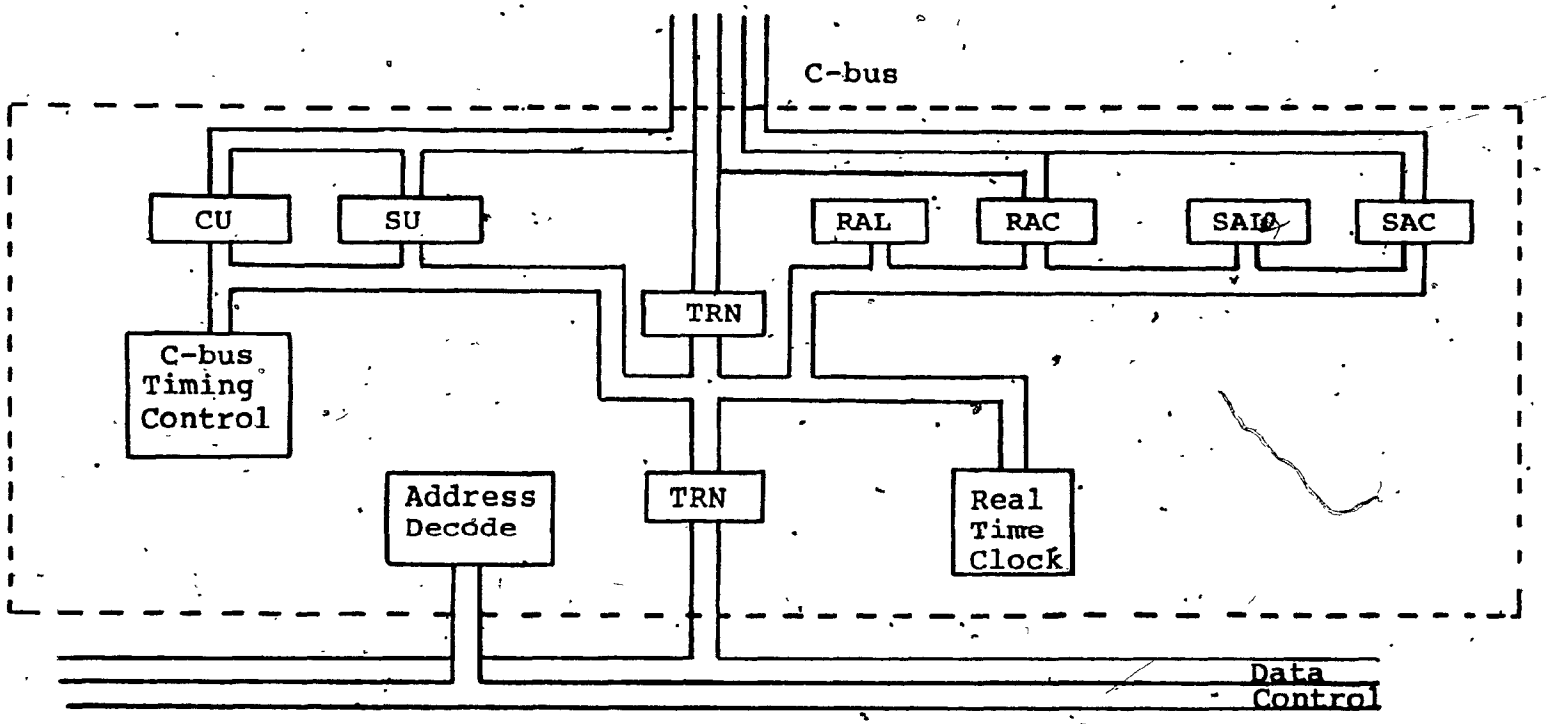
5.2 Hardware Implementation

The C-bus controller has been constructed using an MC6809 single board computer and is augmented with a special purpose logic board. The MC6809 is currently operating at 1 MHZ but its speed will be increased to 2 MHZ upon completion of the debugging phase without any changes required in the special purpose hardware or C-bus interface units. There is space for 4K of ROM and 1K of RAM on the MC6809 processor card which will be used by the C-bus controller software. Upon power up, or reset, the MC6809 will perform an initialization routine and then enter a ready state waiting

for a message transfer request. A set of indicator lamps are provided which when lit indicate the current state of the bus controller. This is used by the operator to monitor the functions of the bus controller.

The special purpose hardware block diagram is shown in Fig. 5.3. The hardware modules are placed into the memory space of the MC6809 and can be accessed by the processor at the addresses given in Table 5.1. In order to control the 16 lines of each address bus, two hardware units are required. The send and receive address latches are used to drive the most significant 8 address lines which remain constant when the bus controller is accessing a specific hardware module on a particular interface. The send and receive address counters are used to drive the least significant 8 address lines. The counters can be loaded with an initial value, incremented by the MC6809, and will be automatically incremented when a data byte transfer between two interfaces is performed. This flexibility is introduced because during a message transfer it is expected that a large succession of addresses will be accessed.

The status, control, and timing control units interact to allow the MC6809 to provide control information and sense the status of C-bus. The status unit occupies several memory bytes, each of which corresponds to one control line that is an input to the bus controller. In addition, the status unit contains a latch through which it can sample and



CU Control Unit
BGRNT Bus Grant
BRST C-bus Reset
CLK Clock
VRA Valid Receive Add.
VSA Valid Send Address

SU Status Unit
BGNTAK Bus Grant Ack.
SLUACK Slave Acknowledge
BREQ Bus Request
PCK Parity Check

RAL Receive Address Latch
RAC Receive Address Counter
SAL Send Address Latch
SAC Send Address Counter
TRN Tranceiver

Figure 5.3

C-bus Controller Special Purpose Hardware

ADDRESS	FUNCTION
F000	Send Address Upper Byte Latch
F001	Send Address Lower Byte Counter
F002	Receive Address Upper Byte Latch
F003	Receive Address Lower Byte Counter
F004	Increment Receive Address
F005	Increment Send Address
F006	Parity Status Bit
F007	Bus Request Status Bit
F008	Bus Grant Issue
F009	Reset Parity Status Bit
F00A	Reset Bus Grant
F00B	Master Computer's Address
F00C	C-bus Data Lines
F00D	Bus Grant Acknowledge Status Bit
F00E	Slave Acknowledge Status Bit
F010	Data Byte Transfer Between Two Interfaces
F011	Read Data Byte From Interface
F012	Write Data Byte To Interface

Table 5.1

Bus Controller Special Purpose
Hardware Address Map

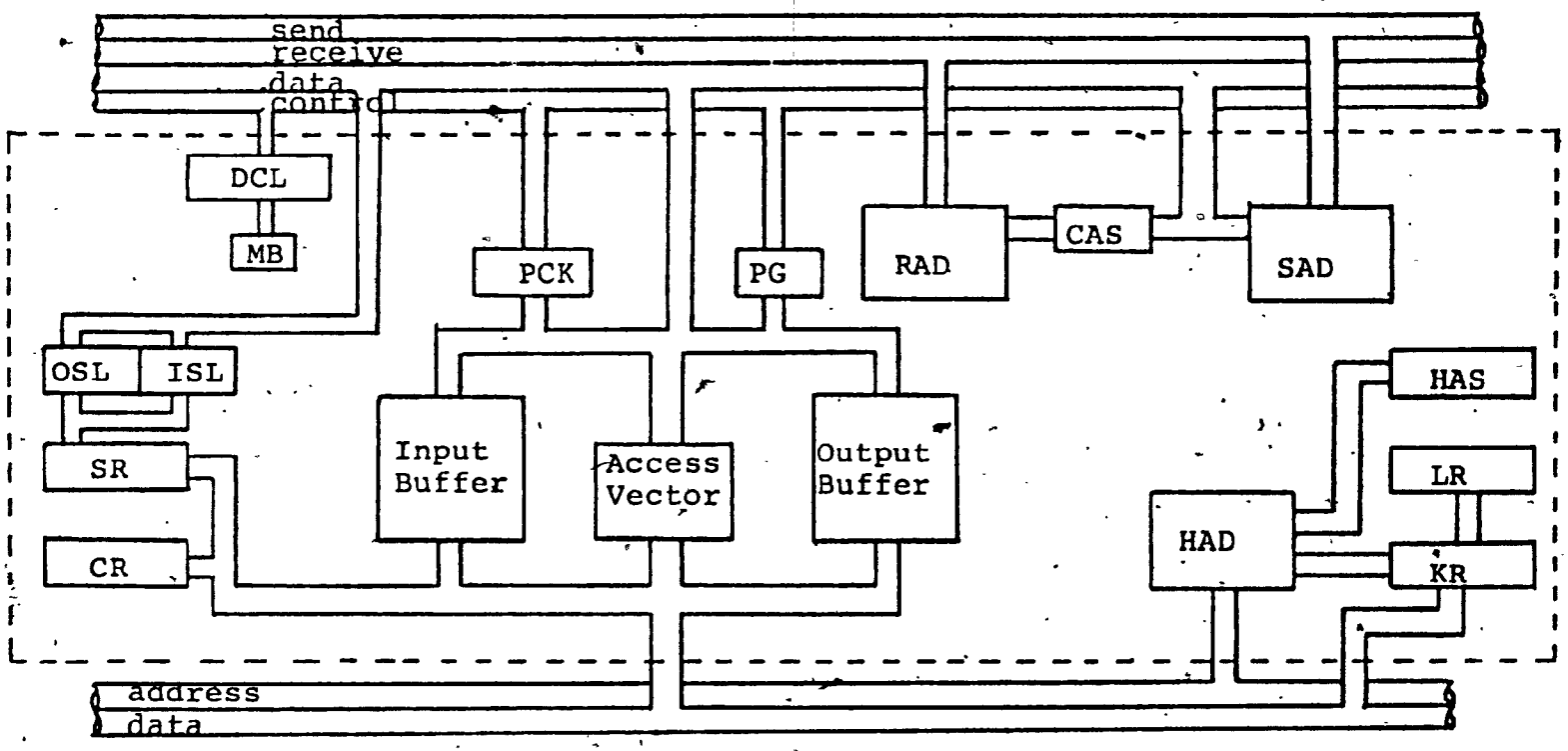
hold the parity information available during an interprocessor byte transfer operation. The control unit is used to drive the control lines that are an output from the C-bus controller. This is performed by the MC6809 in conjunction with the C-bus timing control module. The timing control module is comprised of a number of monostable multivibrators which provide the various timing pulses required during the C-bus read, write, and byte transfer operations. The real time clock module allows the bus controller to provide a time stamp on each message it transfers. The clock will contain a battery back up to maintain the time count during power down conditions.

The MC6809 will be able to interact with the data lines of C-bus through the two transceivers on the special purpose hardware card. In effect, three memory locations act as a gate between the MC6809 data bus and the C-bus data lines. This is accomplished by using the MC6809 memory access cycle to initiate the C-bus timing control module to provide the necessary C-bus timing pulses. Therefore, the MC6809 can interact with the C-bus data lines using its memory reference instructions because no modification of its memory access timing is required. Using the Read Data Byte and Write Data Byte locations, the MC6809 can access the various hardware modules of the interfaces connected to C-bus. The C-bus Data Bus location will also allow the MC6809 to read the C-bus data lines, but the C-bus timing control unit is not activated. This function is used by the controller to

read the address of the interface requesting the service of the bus controller during the selection sequence for the next sender. The C-bus timing control module is not required because it is the responsibility of the interface to control the C-bus data lines during this cycle.

The MC6809 will also be able to activate the C-bus timing control to provide the required control signals to transfer a byte of information between two C-bus interfaces. The information transfer will occur between the two locations selected by the C-bus send and receive address lines. These send and receive drivers must be set to the appropriate value before the timing control unit is activated. The timing control unit will also provide the signals used to sample the parity check signal and increment the send and receive address counters.

The C-bus interface is shown in Fig. 5.4 and the address map for all the modules on the interface is given in Table 5.2. The interface card will simultaneously exist in the address space of its host and the address space of C-bus. The location of the interface card in each address space is controlled by the Host and C-bus address switches. These switches allow for easy relocation of the interface card as the location of each interface will be dependent upon the hardware configuration of the host it is currently connecting to C-bus. All hardware modules that are shared between the host computer and C-bus are isolated by tristate



SR Status Register

B7 Output Buffer
- 0 - Empty
- 1 - Full

B6 Input Buffer
- 0 - Empty
- 1 - Full

CR Control Register

B7 Output Interrupt Cont.
- 0 - Disabled
- 1 - Enabled

B6 Input Interrupt Cont.
- 0 - Disabled
- 1 - Enabled

DCL Daisy Chain Logic
MB Mask Bit
PCK Parity Checker
PG Parity Generator
RAD Receive Add. Decode
SAD Send Address Decode
CAS C-bus Add. Switch
ISL Input Status Latch
OSL Output Status Latch
HAS Host Add. Switch
LR Lock Register
KR Key Register
HAD Host Address Decode

Figure 5.4
C-bus Interface

	Address Bit															
	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Send Address Bus																
Output Buffer	*	*	*	*	*	*	*	0	X	X	X	X	X	X	X	X
Output Bit Reset	*	*	*	*	*	*	*	1	0	0	0	0	0	0	0	0
Mask Bit Set	*	*	*	*	*	*	*	1	0	0	0	0	0	0	0	1
Mask Bit Reset	*	*	*	*	*	*	*	1	0	0	0	0	0	0	1	0
Read Input Bit	*	*	*	*	*	*	*	1	0	0	0	0	0	0	1	1
Receive Address Bus																
Input Buffer	*	*	*	*	*	*	*	0	X	X	X	X	X	X	X	X
Access Vector	*	*	*	*	*	*	*	1	0	X	X	X	X	X	X	X
Set Input Bit	*	*	*	*	*	*	*	1	1	0	0	0	0	0	0	0
Host Address Bus																
Input Buffer	*	*	*	*	*	*	0	0	X	X	X	X	X	X	X	X
Output Buffer	*	*	*	*	*	*	0	1	X	X	X	X	X	X	X	X
Access Vector	*	*	*	*	*	*	1	0	X	X	X	X	X	X	X	X
Output Bit Set	*	*	*	*	*	*	1	1	1	1	0	0	0	0	0	0
Input Bit Reset	*	*	*	*	*	*	1	1	1	1	0	0	0	0	0	1
Status Register	*	*	*	*	*	*	1	1	1	1	0	0	0	0	1	0
Interface C-bus Add.	*	*	*	*	*	*	1	1	1	1	0	0	0	0	1	1
Control Register	*	*	*	*	*	*	1	1	1	1	0	0	0	1	0	0
Key Register	*	*	*	*	*	*	1	1	1	1	0	0	1	0	0	0

* Switch Select
X Don't Care

Table 5.2

C-bus Interface
Address Map

buffers. This will allow the various hardware modules of an interface to be accessed by either the host, or C-bus controller, independent of which other modules are currently in use.

The daisy chain logic module is controlled by the output latch and mask bits. When the output latch indicates that the output buffer is full and the mask bit is not set, the daisy chain logic will activate the bus request line. When the bus grant signal is received by the interface, the daisy chain logic will either place the address of the interface on the C-bus data lines or propagate the bus grant signal to the next interface. The mask bit is controlled by the C-bus controller. The various buffers on the interface have been implemented using high speed static RAM chips. The status register allows the host computer and C-bus controller to sense the state of the input and output buffers and provides synchronization control between the two. The control register allows the host computer to tailor the operations of its message communications software by enabling, or disabling, the interrupt request line for the input and output buffers.

In Fig. 5.5 and Fig. 5.6 we have shown the photographs of the circuit boards that make up the adaptation hardware for the bus controller, and the interface units to C-bus respectively. These circuit boards are presently wire wrapped and hence it takes many man hours to produce one

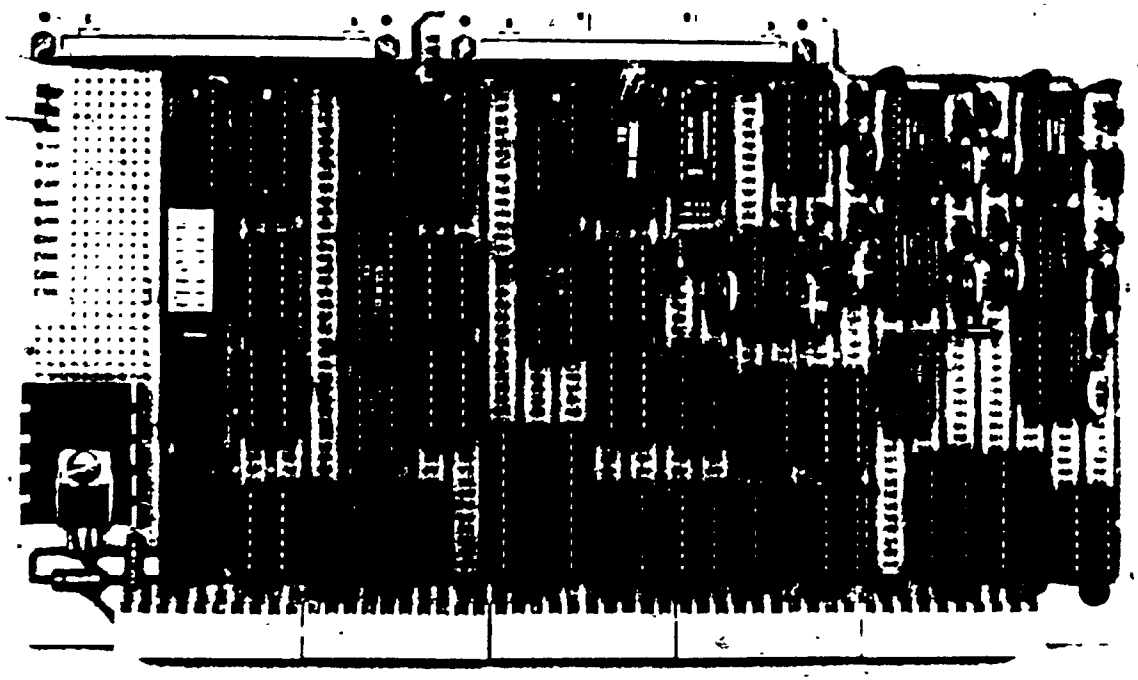


Figure 5.5

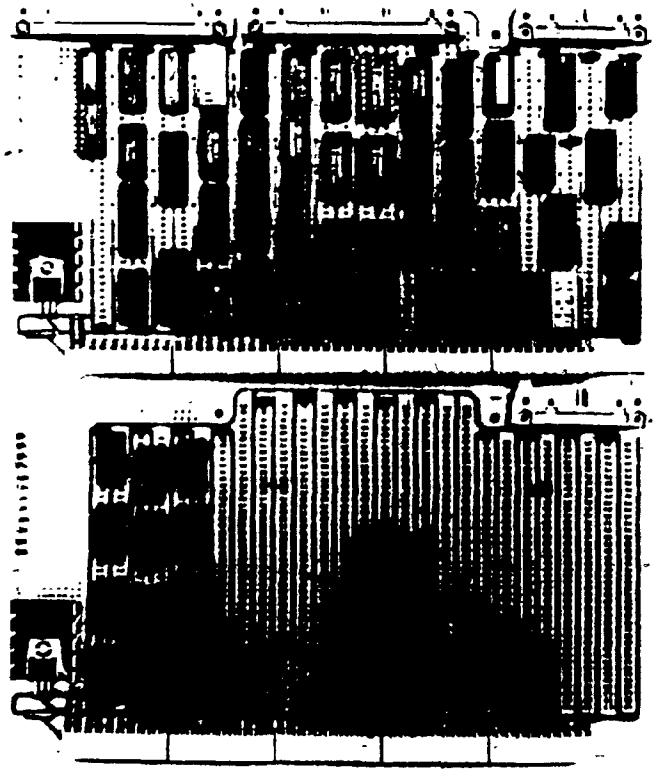


Figure 5.6

set. Each interface can be connected to any of the several T-junctions available on the C-bus through a ribbon cable that plugs into an edge connector on the circuit board. At present, the transmission wires run over a distance of 50 feet and are terminated by a resistive network which results in some attenuation of the signal. Although the C-bus operation is satisfactory, we envisage many improvements in the operational characteristics through the use of trapezoidal line drivers [Balak 82] and better termination. Currently, we have four computers connected to one C-bus. Three of them are MC6809 computers and the fourth one is MC6800.

5.3 C-bus Timing Requirements

There are three data transfer and one control cycle that are needed on C-bus: (a) C-bus controller sends a byte of information to an interface, (b) C-bus controller fetches a byte from an interface, (c) the C-bus controller directs a byte transfer between two interfaces and (d) the selection sequence for the next interface to send a message. These operations are carried out by the C-bus controller with the use of the control lines listed in Table 5.3. Each of these cycles has its own timing constraints which are described in this section.

The selection timing for the next sender is shown in Fig. 5.7. The initial phase of this procedure is

Name	Function
VSA Valid Send Address	The C-bus controller will activate this line to signal to the interface send address decode unit that a valid address is present.
VRA Valid Receive Address	The C-bus controller will activate this line to signal ~~to~~ the interface receive address decode unit that a valid address is present.
CLK Clock	The clock line will be used to enable the interface hardware modules that have been selected during the previous address cycle.
BRST Bus Reset	This line will cause a master reset of all interface hardware modules. All mask bits will be reset during this operation.
BGRNT Bus Grant	The C-bus controller will activate this line to indicate that C-bus is free to send the next message.
BREQ Bus Request	An interface will activate this line when it contains a message to be transfered.
BGNTAK Bus Grant Acknowledge	An interface will activate this line when it receives the bus grant signal and wishes to become the next sender.

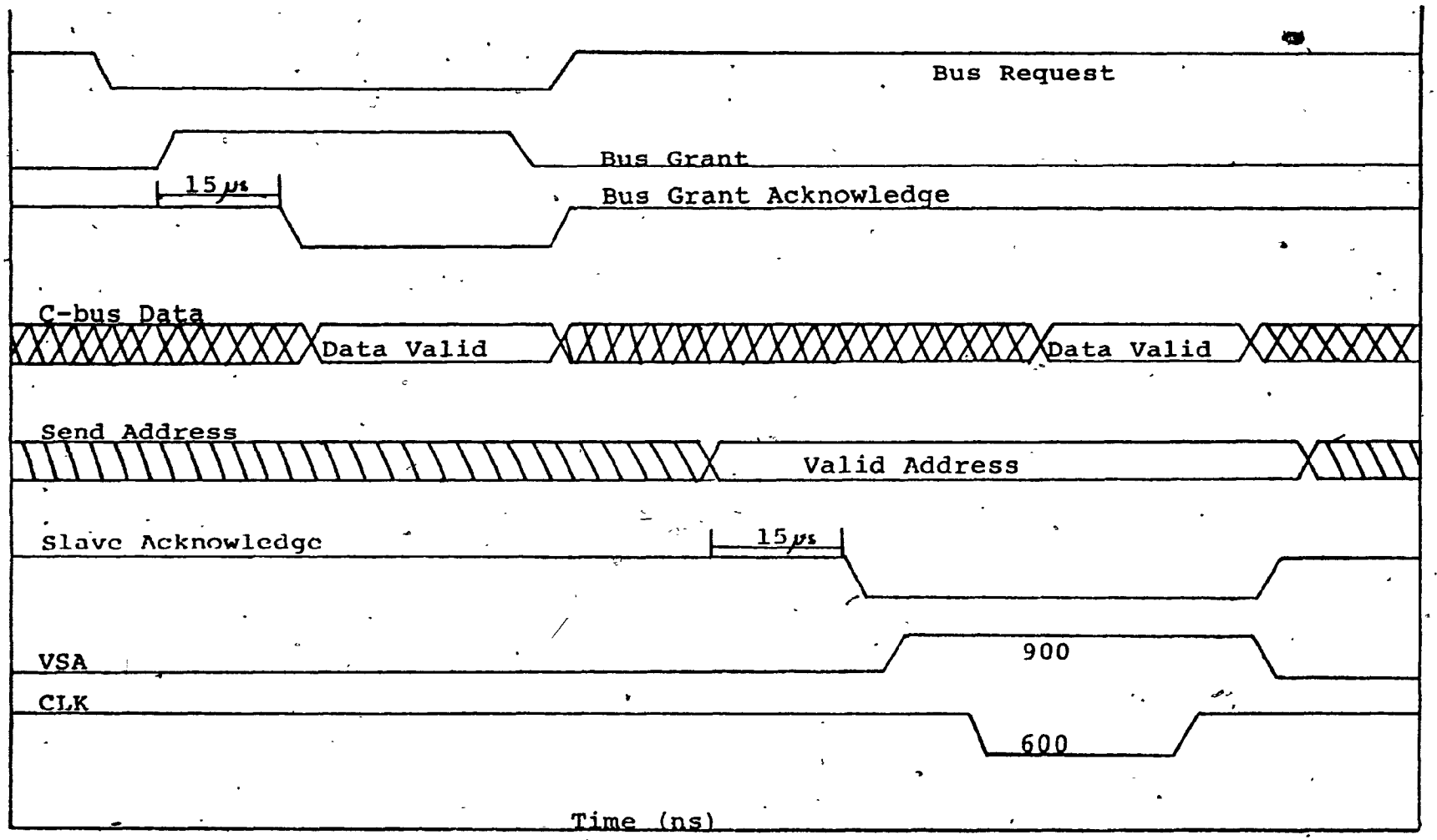
Table 5.3

C-bus Control Lines

Name	Function
SLUACK Slave Acknowledge	An interface will activate this line when its send address decode unit detects that it is being selected.
PCK Parity check	The parity checker will activate this line if a parity error is detected durring a byte transfer operation.

Table 5.3

C-bus Control Lines



Not Valid

Don't Care

Figure 5.7

C-bus Controller Selection Timing

asynchronous and therefore no time values are shown, however there is a specified period after which a time out error will be declared by the bus controller when it is waiting for an acknowledge signal from an interface. Because the initial phase of this selection procedure is asynchronous and uses a handshake protocol, the C-bus controller can sense that a sender has accepted the bus grant signal and that there is an active receiver. Upon sensing a bus request signal, the bus controller will activate the bus grant line when it is free to service the next interface. The C-bus controller will then wait for a bus grant acknowledge signal from the interface which has accepted the bus grant signal and has placed its address on the C-bus data lines. The controller then reads this address on the C-bus data lines, lowers the bus grant line, and the interface will then deactivate the bus grant acknowledge line. The bus controller will then attempt to establish the status of the interface that will be receiving the message by placing the base address (most significant 8 address bits) of the receiving interface on the send address bus and waiting for the slave acknowledge line to be activated. Once the receiving interface acknowledges its address, the bus controller places the complete address for the receive interface's status register on the send address bus. The valid send address line is activated and after an appropriate delay to allow for the address selection logic to stabilize, the clock line is activated to enable the

drivers on the receiving interface to place the value contained by the status register onto the C-bus data lines.

The timing of a byte transfer from an interface to the bus controller is shown in Fig. 5.8 and the timing of a byte transfer from the controller to an interface is shown in Fig. 5.9. The bus controller sets the send, or receive, address latch and counter to the appropriate values. In the next cycle a memory load, or store, instruction is executed using the address which will activate the C-bus timing control unit to provide the timing pulses to perform the send, or receive, operation. The valid send, or receive address, line will become active and time is allowed for settling of the address lines. Then CLK will become active to enable the interface hardware module that has been selected to place information onto, or read, the C-bus data lines. The MC6809 will either read from, or write onto, its own data lines which for this operation are connected to the C-bus data lines.

The timing requirements of C-bus for a byte transfer between two interface cards is shown in Fig. 5.10. The MC6809 of the C-bus controller will set up the send and receive address latches and counters to select the send and receive interface. In the next cycle, the MC6809 will activate the C-bus timing control module to produce the desired timing signals. The valid send and receive address lines are activated and the address decode modules on the

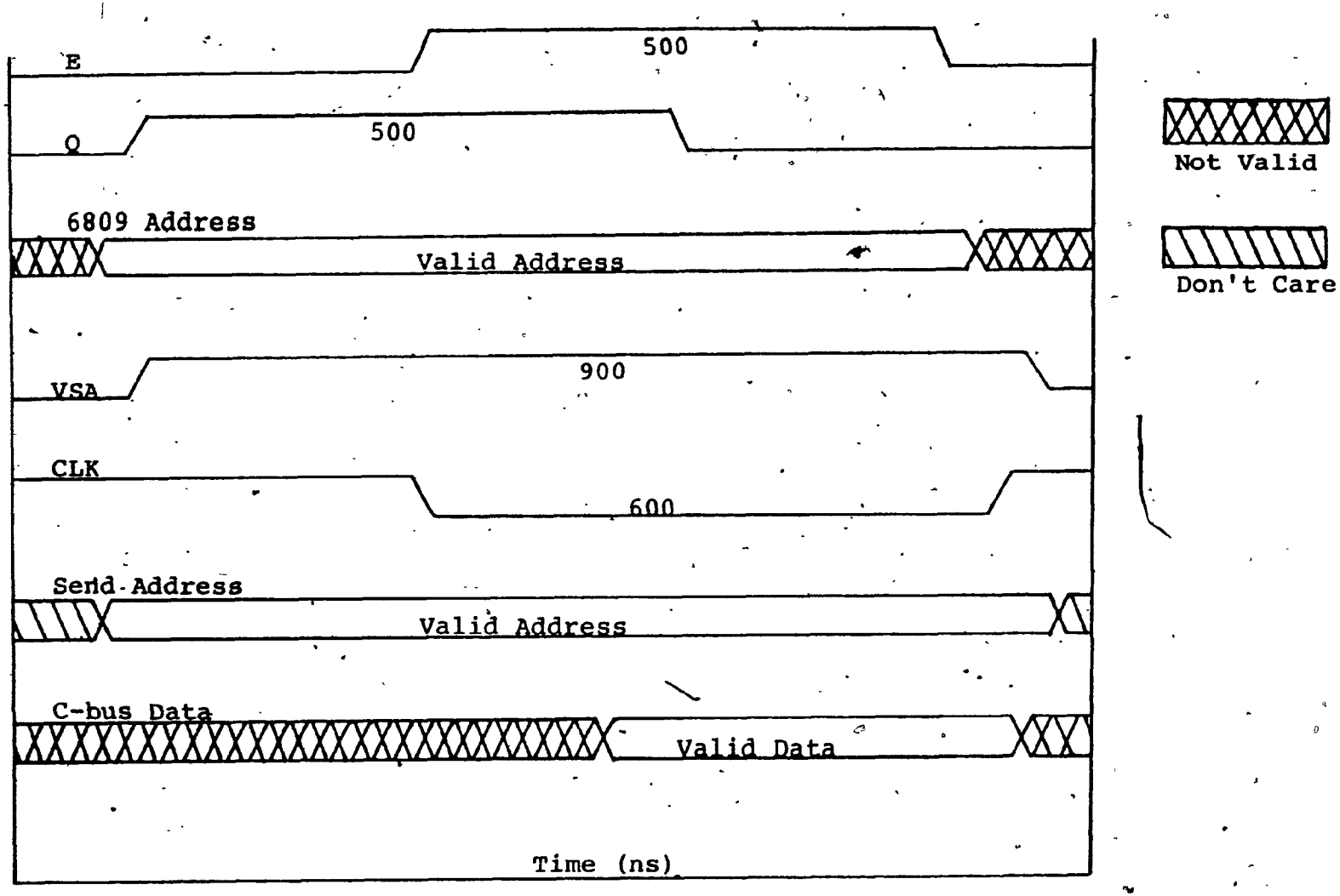


Figure 5.8
6809 Control of a Byte Transfer
From a Computer to the C-bus Controller

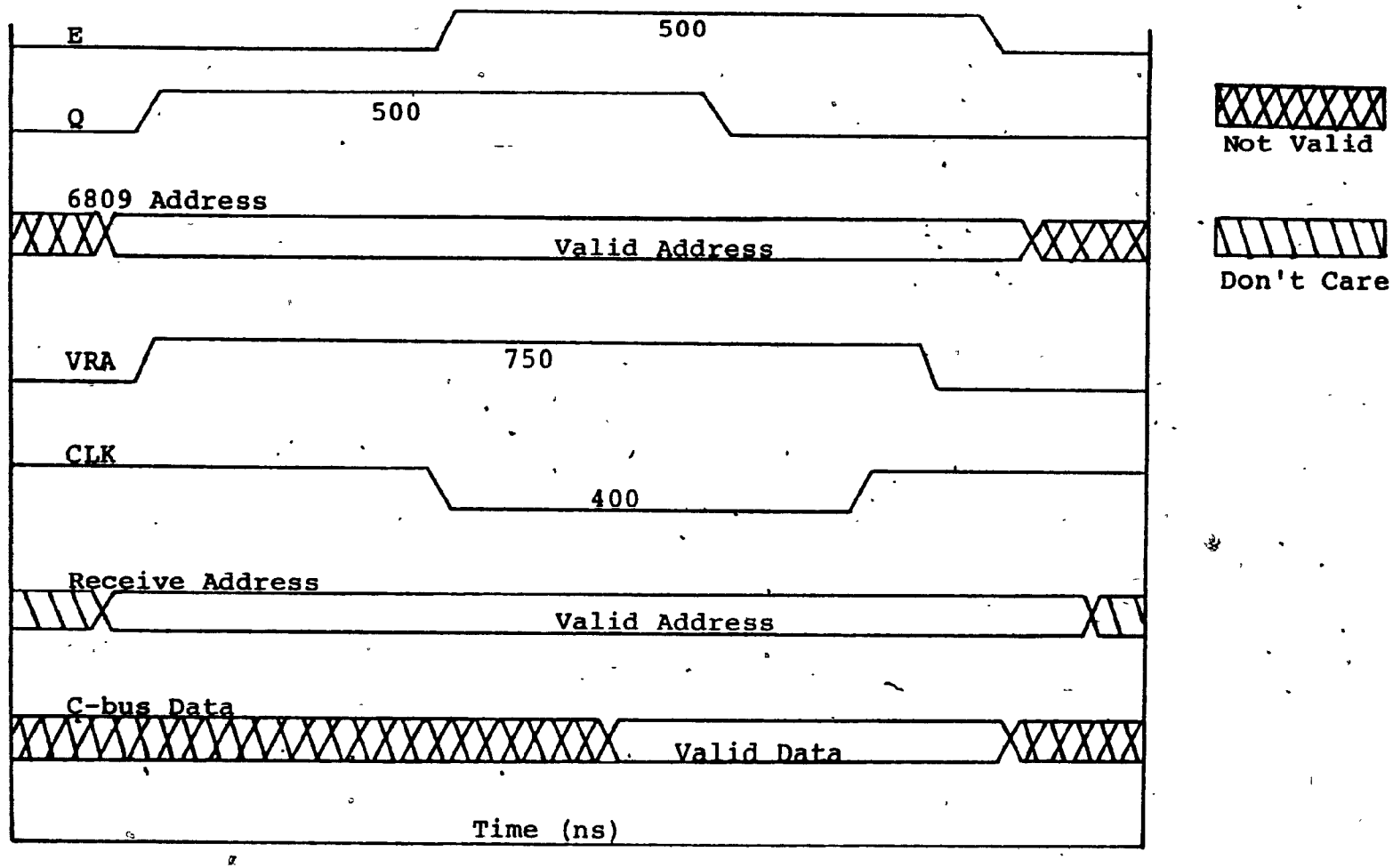


Figure 5.9
6809 Control of a Byte Transfer
From the C-bus Controller to a Computer

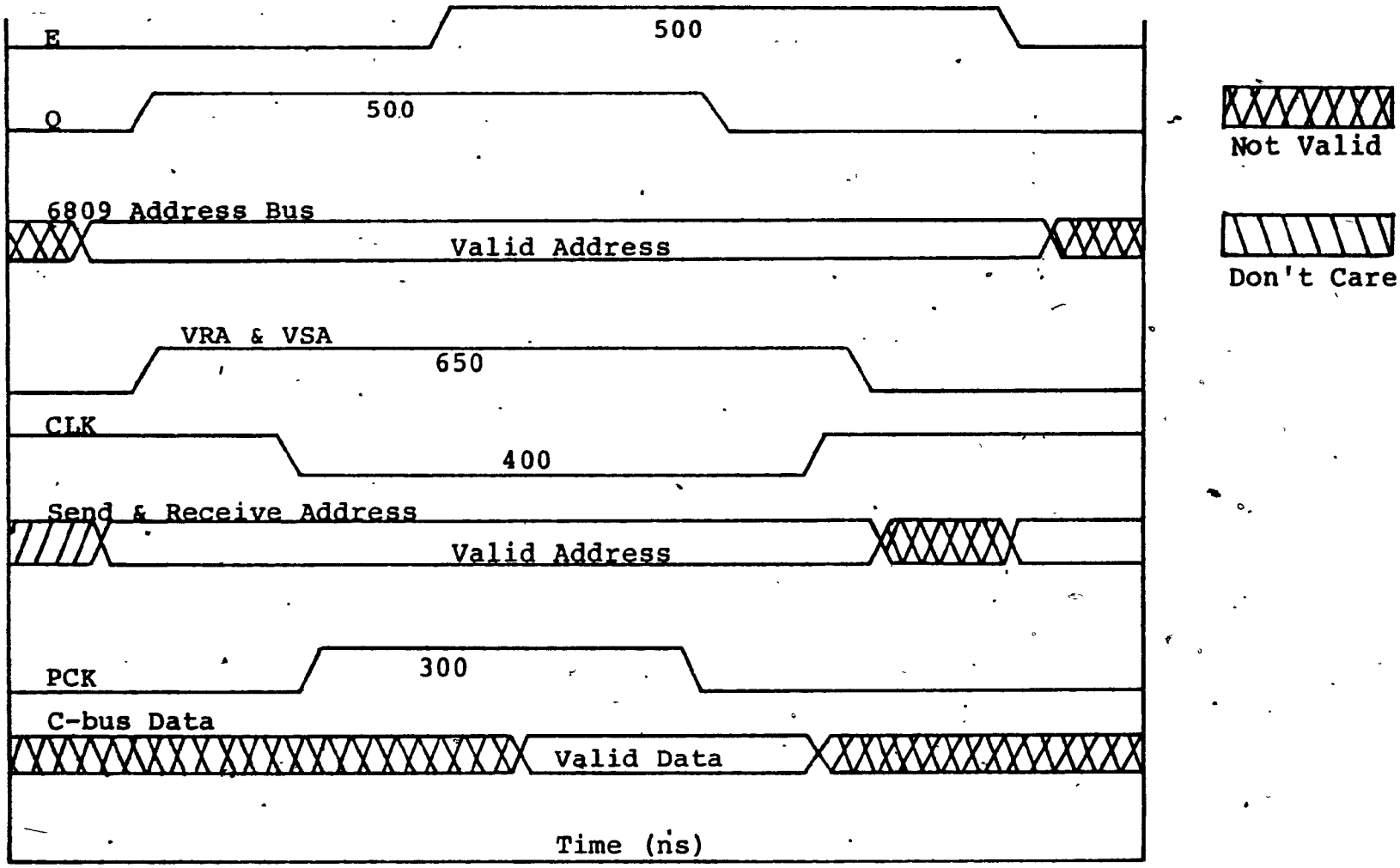


Figure 5.10

6809 Control of a Byte Transfer
Between Two Computers on C-bus

send and receive interface are allowed 175 nanoseconds deskew time. At this point the clock signal is activated which will enable the sending interface to place a data byte onto the C-bus data lines, allow time for settling of the data lines, and the receiving interface to read the data lines. In addition, the output of the parity generator of the receiving interface is sampled during the period when the C-bus data lines contain valid information. After the valid address lines and the clock signal are deactivated, the send and receive address counters are incremented in order to prepare for the next byte transfer. This operation requires 150 nanoseconds for date line deskew.

The length of the byte transfer operation will determine the maximum transfer rate of C-bus. This will be dependent upon the speed of the memories used to implement the input and output buffers of the interfaces and the length of the C-bus cable. In the present prototype, the time required to transfer one byte of information is approximately one microsecond. This would indicate that the maximum transfer rate of the C-bus prototype is 1 M Bytes per second. However, because the C-bus controller is based on a general purpose microprocessor, this speed is not achieved. The final speed of the C-bus prototype will depend upon the cycle time of the microprocessor and the efficiency of the software algorithms devised to run the bus controller which are discussed in the following section.

5.4 C-bus Controller Software

At present, the C-bus controller can be driven by two separate programs. A debug program has been written which will accept commands from a terminal and perform various operations that will simplify the checking and installation of an interface card. A C-bus driver has also been written which will perform all the functions required to enable the microprocessor of the C-bus controller to interact with the special purpose hardware. The design of the C-bus driver will be critical in determining the final speed of C-bus. Because the microprocessor used in the C-bus controller is slow compared to the transfer delay caused by C-bus itself, all operations of the C-bus controller must be performed as efficiently as possible. It is expected that the debug package will run many times slower than the C-bus driver and will not be used under normal load conditions.

The debug package will accept the list of commands shown in Table 5.4. The first three commands allow the operator to test the input, output buffer, and access vector. The next two commands offer a lower level of verification where a specific address is continuously enabled. This is useful once a problem with a specific hardware module has been identified and a technician wishes to investigate further using an oscilliscope. The next two commands will essentially perform the same functions as the C-bus drivers. However, a full trace of the message

Command	Function
I##	Load the input buffer of the interface whose base address is ## with a test pattern.
O##	Read the output buffer of the interface whose base address is ## and display the contents at the terminal.
A##	Load the access vector of the interface whose base address is ## with a test pattern.
S#### (CR)-EXIT	Continuously send a byte to the C-bus interface hardware module at address ####.
R#### (CR)-EXIT	Continuously read a byte from the C-bus interface hardware module at address ####.
G (CR)-EXIT	Enter a message send mode. The C-bus controller will now be actively sending messages between the computers of CUENET upon request. All the actions of the bus controller, including error conditions, are reported to the terminal. The C-bus controller will also halt after each message transfer.
C (CR)-EXIT	Same as G command but the controller does not halt after each message.
Q	Quit.

#-One hex digit

Table 5.4

C-bus Debug Commands

requests received, and processed, is displayed at a terminal attached to the bus controller. In the "G" mode the C-bus controller will stop after each message to allow for stepwise debugging of the CUENET operating system and to monitor the flow of messages on C-bus. In the "C" mode the C-bus controller is not halted between each message. In the future it will be possible to have the C-bus controller enter a debug mode under control of the CUENET master and all trace information would also be relayed to the master. This would allow the CUENET operating system to perform tests on the hardware when a fault is detected and report to the operator if outside assistance is required.

The C-bus driver will operate without the need for outside assistance such as commands input from a terminal. In the case that the C-bus controller detects an error it will prepare a special message to notify the CUENET master. At present, there are three types of messages which the C-bus controller will consider valid: (a) an interprocessor message which would include the type I and type II messages described in the simulation chapter, (b) a request by the CUENET master to load an access vector, (c) and an error message transmitted from the C-bus controller to the CUENET master. The format of an error message and a list of the valid error codes is given in Table 5.5. It is the responsibility of the CUENET master to take the necessary actions to recover from any error recognized by the C-bus controller. As noted previously, C-bus interface contains a

Error Message Format

Byte	Function
00	Sender Address
01	Receive Address
02	Message Length
03	Message Type
04	Time Stamp
05	Time Stamp
06	Time Stamp
07	Message Origin that caused the error
08	Destination of the message that caused the error
09	Error code
10	End of Message

Error Codes

Error Code	Error
01	No bus grant acknowledge signal was detected in the specified time period.
02	A message request was received with an invalid message code.
03	A parity error was detected on several attempts to transfer a message.
04	A computer other than the CUENET master requested the C-bus controller to load an access vector.
05	No slave acknowledge signal was detected within the specified time period.

Table 5.5

Error Message Format and Error Codes

mask bit which can be used by the controller driver to establish a priority in the daisy chain. The C-bus controller driver uses a software stack to establish a round robin [Liste 79] priority scheme. Each message request that cannot be processed because the destination interfaces input buffer is busy will be placed onto a stack. When there are no current requests, the C-bus controller will empty the stack and then process those requests. It should be noted in times of heavy bus usage the requests that are stacked may subject to large delays. This problem can be handled by also placing a time limit each message request may spend on the stack.

The flow diagram for the operations of the C-bus driver is given in Fig. 5.11 and a listing of the software can be found in Appendix II. In the current version of the C-bus driver, we have been able to estimate the speed of C-bus by counting the number of machine cycles required to process each message. This can be further broken down into the overhead cost of each message plus the time required to transfer each byte. The loop used to transfer each byte on C-bus requires 9 machine cycles, or 4.5 microseconds, even though, as mentioned in section 5.3, C-bus requires only one microsecond for the actual transfer. The overhead cost for each message, provided no errors are detected or retries are required, is approximately 200 machine cycles, or 100 microseconds. If we combine these two costs and assume a message size of 248 data bytes, we arrive at a transfer rate

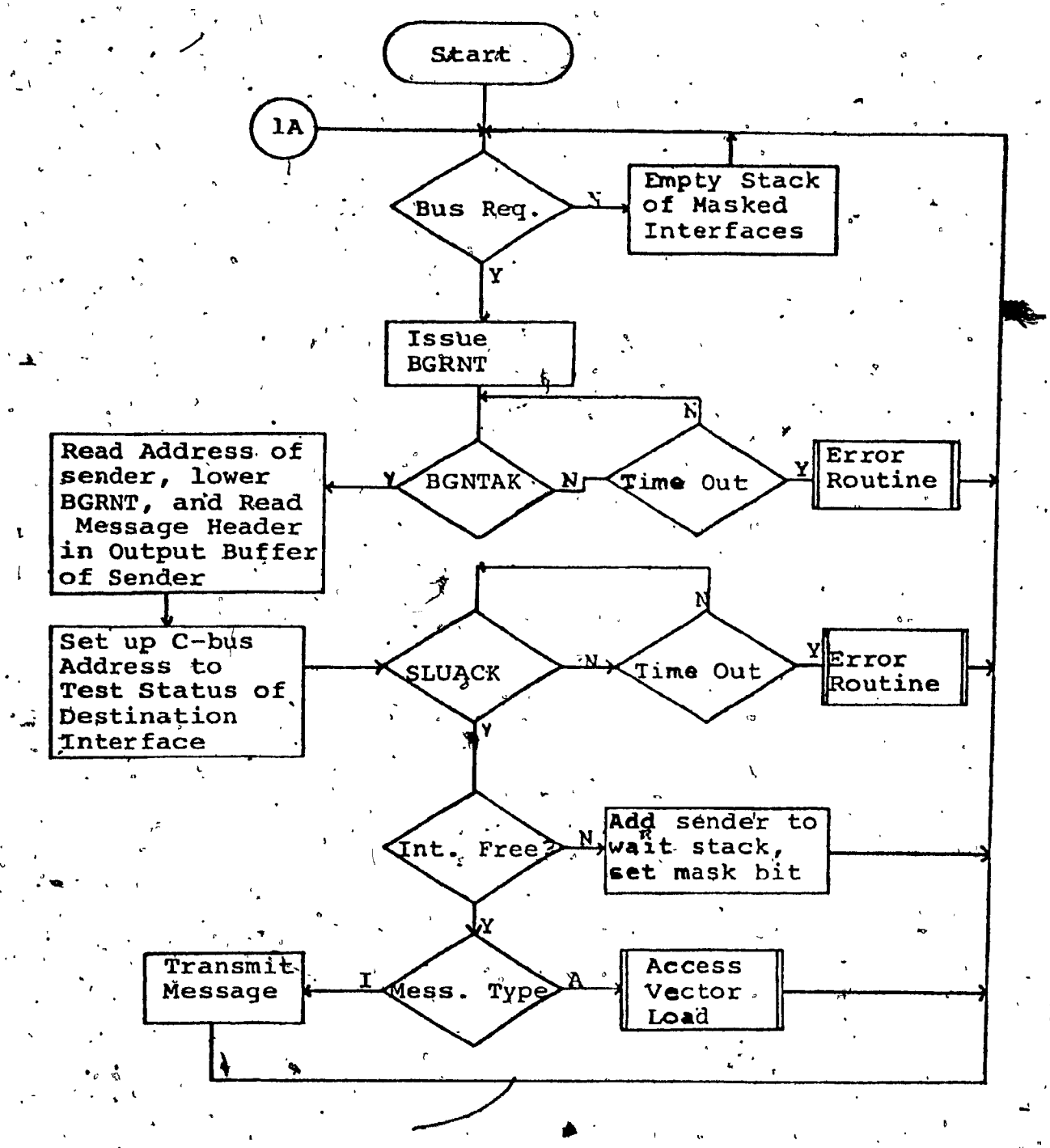


Figure 5.11

C-bus Control Program
Flow Chart

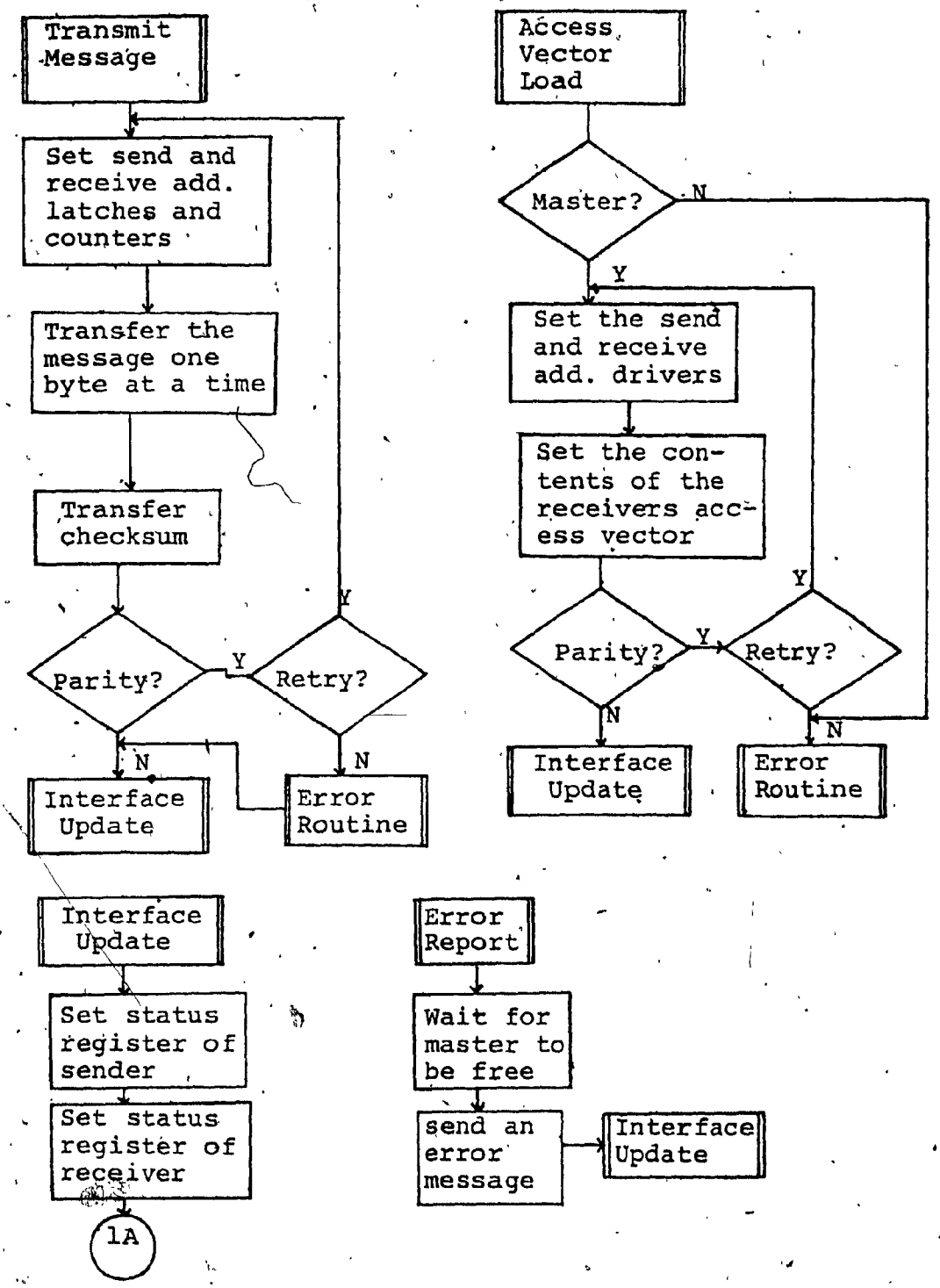


Figure 5.11
C-bus Control Program
Flow Chart

of 1.6 mega bits per second that can be achieved for our prototype with the speed of the current C-bus controller. Of course, as the speed of the C-bus controller is increased by increasing the clock rate of the microprocessor, or using a faster technology, greater transfer rates can be achieved without any modifications to the other components of C-bus.

CHAPTER VI

APPLICATIONS AND FUTURE DEVELOPMENT

6.1 Evaluation of Decompositions

In order to determine the effectiveness of a particular decomposition we would normally compare the expected speed of the parallel algorithm with the time required to execute the original sequential algorithm on a uniprocessor. The expected speed of a decomposition can be estimated as the sum of the theoretically best possible speed up and the overhead incurred due to multiprocessing. For a given decomposition, this overhead will be affected by many factors, some of which are dependent on the architecture chosen, and the characteristics of the multiprocessor used, while others are determined by the implementation of the various tasks of the decomposition. These parameters are as follows:

(Vl*Tl): This product is a measure, in units of time, of the overhead due to code loading that is required in order to initialize an architectural configuration. Vl represents the amount of program code to be loaded, and Tl is the effective time required to load a unit of code.

(Vc*Tc): Will give us a measure of the time overhead due to data transfers between the tasks of a division. Vc is the amount of data to be transferred between tasks, and Tc represents the effective time to transfer a unit of data between any two tasks.

(Vd*Td): This product is a measure, in units of time, of the unused processing power of the processors during periods when they must wait in order to obtain access to a shared resource, or for the purpose of process synchronization. Vd represents the number of times a task will need to access a critical region or wait for a synchronization signal from another task. Td will be the average time wasted for each time a task must access a critical region or perform a synchronization operation.

The parameters Tc and Tl will depend upon the characteristics of the multiprocessor upon which the decomposition will be executed. In the case of CUENET Tc will be determined by the effective transfer rate of C-bus while Tl will vary depending upon which computer of CUENET will be used to execute a particular task and where in the network the code resides. The remaining parameters will depend upon the particular decomposition because they are affected by the characteristics of each task and the topology of interactions between the various tasks of a decomposition.

Generally, we wish to minimize the three products, (Vl*Tl), (Vc*Tc), and (Vd*Td). When they increase, the effective data throughput of a decomposition decreases accordingly. We can make use of the parameter (Vl*Tl) to give us an idea of the time needed to configure the system so the desired decomposition may be executed. Of course, this set up time should be small compared to the overall execution time of the decomposition. In the performance

measure of an optimized multiprocessor system, we may assign different weights for the throughput and resource utilization parameters. Two decompositions may be compared with respect to their expected throughput and the amount of anticipated unfruitful processor time. For a system such as CUENET we would tend to assign a higher weight to data throughput, or speedup, than to resource utilization, because the cost of the microprocessors and memory modules are not overly significant.

The Vd parameter can be expressed as a vector vdi i=1,2...n, where each vdi represents the expected number of accesses to shared resource i by all processors that are allowed to use that resource. This parameter can be further described as

$$vdi = \sum_{1}^{m} vij$$

where:

vij: Is the number of expected requests by processor j for the use of shared resource i

m: Is the number of processors

If we define Td as tdi i=1,2...n where

tdi: Is the amount of expected delay, for a request to use resource i, that a processor will encounter.

then in the worst case, where all the resource access delays occur sequentially (there is no overlap of the waiting times of any of the processors), we can evaluate the product

$$(Vd*Td) = \sum_{1}^{n} vdi*tdi$$

However, one must realize that in most cases the worst case

value is too pessimistic. The value computed can still be useful in the following context:

(a) It will give the designer an idea of the upper bound to which the throughput of his decomposition can, or will, be degraded due to this type of delay.

(b) The designer can sometimes make assumptions about the amount of overlap expected from the data set, or from observation of the sequential algorithm. Secondly, even if it is difficult to estimate the degree of overlap, the system designer can assume a certain amount of overlap, which can be used for the comparison of two decompositions.

In order for a software analyst to decide between various possible decompositions, the effect of each of these decompositions on the various overhead parameters must be evaluated. In many cases it will not be possible to find an exact value for these parameters because of the dynamic nature of interactions between the tasks of a decomposition. Therefore, the parameter values will have to be measured by execution of the algorithm upon CUENET or from measurements done on portions of the original sequential algorithm running on a uniprocessor.

6.2 Current Applications on CUENET

As noted in [Fancot 80], partitioning a user job into several coordinated tasks for the purposes of concurrent execution is still an open problem. In the use of CUENET, we anticipate the user to contribute in arrving at such partitioning. Consequently, we need a multiprocessor

language in which the user can easily specify the possible concurrent executions and the necessary precedence constraints, or synchronizations, among the concurrently executed tasks. We are working on an extension to Edison, the new multiprocessor language developed by Per Brinch Hansen [Hanse 81]. In the extended Edison, a user can specify the number of processors, the types of processors, and the topology of interconnection between them that he requires to solve his problem.

In one of the application oriented software projects, we are using CUENET as a local area network for the implementation of an office information system. We have found that the alternative based language called ABL, to be well suited for describing an office environment [Leben 81]. This language has a potential to express the parallelism in a problem explicitly. Also, an interpreter for ABL written in Pascal is readily available to us. In the proposed system each computer connected to the CUENET will function as a work station of an office. Interstation message communication will be carried out over the C-bus. The work station software, written mainly in ABL, will be responsible for local processing.

An operating system is an essential part of a computer system. Two operating systems, namely Star OS and Medusa, have been written for Cm* [Jones 79], and the operating system MICROS [Witti 80] has been developed for MICRONET.

At least at the initial stages we are not attempting to develop a completely distributed operating system for CUENET. Instead, a single machine operating system will be augmented by an additional layer of software to account for the functions explained in chapter 3. Currently, the master software is being developed as an extension to the FLEX operating system of MC6809 microcomputers [Flex 80]. We are also planning the development of the master software as an extension to the popular CP/M operating system [Murth 80].

In another application of CUENET, we make use of its parallel processing power for linear predictive analysis (LPC) of speech signals. Analysis of speech signals in real time to extract the LPC parameters requires a substantial amount of processing power that is not available in one single microprocessor. A complete LPC analysis involves several stages of processing such as data acquisition, calculation of predictor coefficients, pitch extraction, gain calculation, and voiced or unvoiced decisions [Makou 75]. These operations must be repeated for every ten milliseconds of speech. A pipeline architecture could be used for LPC analysis where each stage of the pipeline will perform some of the above mentioned computations. When the computations of different stages are arranged properly, it is possible to minimize the time loss due to message transmission so that the different computers of CUENET can be used to perform the computations concurrently and achieve a measurable speed up [Seeth 82].

6.3 Future Hardware and Software Development

At present, we have four C-bus interfaces that are operational. CUENET is comprised of three computers, two motorola MC6809's and one MC6800. One MC6809 computer system is now being used as a C-bus controller. A special purpose dedicated single board computer will be constructed to act as a C-bus controller. The MC6809 computer system that is now acting as a C-bus controller will then be added to CUENET using the fourth C-bus interface. A fifth C-bus interface has been designed for an MC68000 and will shortly be wired. Then CUENET will contain five computers where the master will be the MC68000, the three MC6809 computers will be slaves, and the MC6800 will be used as a network memory unit.

This C-bus prototype will be limited in speed by the microprocessor based C-bus controller. One possible extension to the current network would be to construct a bit slice based controller, as described in Chapter 5, which would now be feasible since the protocols chosen for the microprocessor based C-bus controller have been shown to work. The simulations described in Chapter 4 indicates that the speed of C-bus will be greatly enhanced by using such a controller. The present C-bus prototype uses standard TTL bus drivers which are only reliable for transmission of

digital signals over short distances but by switching to trapedoidal type line drivers [Balak 82] and receivers the length of C-bus could be significantly extended.

In order to aid in the evaluation of a particular decomposition executing on CUENET it would be very convenient if we could make measurements on the variety and volume of messages that are generated by each computer. The results obtained from these measurements would then be displayed to the user at the end of each run of his job. If we were to attempt to implement this through software routines in each computer and the C-bus controller, the transfer rate of C-bus would be seriously degraded. We could add additional hardware registers and counters to each C-bus interface which would automatically record various statistics about the messages received. At the termination of a user algorithm, the CUENET operating system could gather the information available at each C-bus interface and present it to the user.

Currently, the device drivers that are required to transmit messages over C-bus have been written and tested along with the C-bus controller programs described in Chapter 5. An operating system can now be designed to handle the problems that must be faced because of the multicomputer environment now that the CUENET hardware structure has been defined and tested. At first, a multicomputer layer will be added to the Flex operating

system, which will enable the 6809 computer to act as CUENET slaves. The software for the network memory unit will have to be written as a stand alone program because only a primitive monitor is available for the 6800. Finally, the major programming effort will be to create the CUENET master software which will be targeted for the 68000 computer system.

The CUENET slave software will be able to interpret and execute various directives that it receives from the master. These directives will instruct a CUENET slave to perform operations such as load a program from its own mass storage, accept and save a program send to it over C-bus, start execution of a specific algorithm, terminate an algorithm, and respond to status requests. If the slave software traps an error of any kind it will notify the CUENET master of the condition and wait for further instructions from the master. These errors will include any errors related to the operation of C-bus such as a request by a user algorithm to send a message to a computer system for which it does not have access rights.

The CUENET master software will be responsible for interfacing with the users and coordinating the operations of all the slave computers and network memory units. The master will have to maintain information on the resources and status of all the computer systems connected to CUENET. This information will also be used for allocation of user

algorithms to the various slave computers. This allocation procedure will have to be based upon the resource requests made by the user and the resources available. Upon termination of a user algorithm, the master will be responsible for gathering all measurements made by the C-bus interfaces and prepare them for presentation to the user. All errors that are encountered by the slave computers and the C-bus controller must be recognized by the CUENET master and recovery from these errors will be the responsibility of the master.

A programming language or operating system for CUENET will have to include a mechanism through which the user can make use of the reconfigurability permitted by C-bus. Essentially a language structure must be defined which will allow a user to define the characteristics of each computer that is required to execute one of the tasks of his decomposition and the access patterns that are expected among all the CUENET slaves being used to execute his algorithms. The code generated by such a computer would have to be a set of modules, each of which could be loaded into a slave and executed. Along with this module, some identification information will be required to keep track of which module corresponds to each task of a decomposition and the resources each module requires as specified by the user. This information will be required during the slave allocation and loading process. The developement of this software is currently being carried out as a separate

masters thesis.

6.4 Conclusion

We have fabricated a single CUENET prototype which contains five computers: one master, three slaves, and one NMU. At the beginning of this project the resources at our disposal were one MC6800 microcomputer with no mass storage devices. A monitor program was writen which would allow us to down load programs from a mainframe computer. This microcomputer has now become our network memory unit. We then purchased an MC6809 microcomputer with 56K RAM. For mass storage we purchased a 10MByte fixed drive and an eight inch floppy drive from seperate manufacturers. The fixed and floppy drives were then installed in a single cabinet with a common power supply. The device drivers of the operating system that was available for the MC6809 had to be modified so that it could function with this type of mass storage device because it had been designed to operate with a different type of disk controller by its original authors. Once this prototype system was operational we then proceeded to produce a second version. These two systems are now used as slave computers for CUENET. The third CUENET slave is also an MC6809 computer which was available within this department but required some repairs to its power supply before it was operational. The CUENET master is an MC68000 microcomputer which has been purchased as a complete system with a 20MByte fixed and eight inch floppy drives. We are

presently installing and testing this system. The C-bus cable was installed and the appropriate termination network constructed. At this point we were ready to build a prototype of the C-bus interface and the special purpose hardware for the C-bus controller. One prototype for each board was wire wrapped and tested. Once our design was shown to work, three more C-bus interfaces were assembled. A fifth C-bus interface is currently under construction which is designed for the multibus and will be used by the CUENET master. A single board MC6809 computer and power supply have been assembled to be used as the C-bus controller when the C-bus prototype reaches its final stage. We have also designed and fabricated an interval timer calendar clock interface which will be useful for statistical measurements. Finally, the necessary software drivers for the C-bus controller and communications layer of the CUENET operating system have been written.

Throughout this project we have followed an engineering approach to our design methodology. We assessed our immediate requirements and resources and then performed an analysis, or simulation to determine if our design would be adequate. Only when the preliminary analysis showed positive results did we attempt to implement our paper design. At each stage in the project the components under construction were thoroughly tested by tests which had been designed during the analysis stage. We now have an interactive debugger used by the C-bus controller for

testing all the interface units connected to C-bus. A second thesis is now in progress which will complete the CUENET operating system.

We have built the foundation for the CUENET prototype. This has made it possible to conduct further research in parallel processing, in the areas of operating systems and applications software, and in the applications of local area networks to office automation. The major contributions of this thesis are the novelties in the design of CUENET, and C-bus, and the engineering approach we adopted in the design, construction, and testing of our prototype.

The implementation of a project such as this in a normal university environment with limited resources involves many problems. While the solutions to these problems may be technically trivial the total amount of time and effort spent on such tasks are significant.

REFERENCES

[Adams 78] G. Adams, and T. Rolander. Design Motovations for Multiple Processor Microcomputer Systems. Computer Design, March 1978, pp. 81-89.

[Adams 82] G.B. Adams, and H.J. Siegel. The Extra Stage Cube; A Fault Tolerant Interconnection Network for Supersystems. IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 443-454.

[Ahmed 82] H.M. Ahmed, J.M. Delosme, and M. Morf. Highly Concurrent Computing Stuctures for Matrix Arithmetic and Signal Processing. IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 65-82.

[Akkoy 74] E. Akkoyunlu, A. Bernstein, and R. Schantz. Interprocess Communication Facilities for Network Operating Systems. IEEE Computer, June 1974, pp. 46-55.

[Andre 80] F. Andre, J.P. Banatre, H. Leroy, G. Paget, F. Ployette, and J.P. Routeau. KENSUR: An Architecture Oriented Towards Programming Language Translation. Seventh Annual Symposium on Computer Architecture 1980, pp. 17-22.

[Arden 82] B.W. Arden, and R. Ginesar. MP/C: A Multiprocessor Computer Architecture. IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 455-473.

[Arnol 82] R.G. Arnold, R.O. Berg, and J.W. Thomas. A Modular Approach to Real-Time Supersystems. Vol. C-31, No. 5, May 1982, pp. 385-398.

[Arulp 80] J.A. Arulpragasam, R.A. Giggi, R.F. Lary, D.T. Sullivan, and C.C. Wu. Modular Minicomputers using Microprocessors. IEEE Trans. on Computers, Vol. C-29, No. 2, 1980.

[Baer 80] J.L. Baer. Computer Systems Architecture. Computer Science Press, 1980.

[Balak 82] R.V. Balakrishnan. Eliminating Crosstalk Over Long Distance Bussing. Computer Design, March 1982, pp.155-162.

[Barne 68] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stockes. The Illiac IV Computer, IEEE Trans. on Computers. Vol. C-17, No. 8, 1968, pp. 746-757.

[Batch 82] K.E. Batcher. Bit-Serial Parallel Processing Systems. IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 377-384.

[Bell 71] C.G. Bell, and A. Newell. Computer Structures Readings and Examples, McGraw-Hill, New York, 1971.

[Berg 72] R.D. Berg et al. Pepe: An Overview of Architecure, Operation and Implementation. Proc. of Nat. Electronics Conf., 1972, pp. 312-317.

[Buchb 79] B. Buchberger, J. Fergerl, and F. Lichtenberger. Computer-Trees: A Multicomputer Concept for Special Purpose Parallel Processing. Microprocessors and Microsystems, Vol. 3, No. 6, July 1979.

[Cambr 80] ------ Cambridge Digital Communication Ring. Computer Laboratory, Cambridge University, U.K., 1980.

[Civer 82] P. Civera, G. Conte, D. Del Corso, F. Gregoretti, and E. Pasero. The U* Project: An Experience with a Multimicroprocessor System. IEEE Micro, Vol. 2, No. 2, May 1982, pp. 38-50.

[Dewit 79] D.J. Dewitt. DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems. IEEE Trans. on Computers, Vol. C-28, No. 6, June 1979.

[Enslo 77] P.H. Enslow. Multiprocessor Organization: A Survey. Computing Surveys, Vol. 9, No. 1, 1977, pp. 103-129.

[Fairb 82] D.G. Fairbairn. VLSI A New Frontier for Systems Designers. IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 87-96.

[Fanco 80] T. Fancott. A Communications Process for a Distributed Multiprocessor Operating System. Proc. of the Canadian Comm. and Power Conf., Montreal, Oct. 1980,.

[Flex 80] --------. Flex Users Manual and Advanced Programmers Guide. Technical Systems Consultants Inc., 1980.

[Flynn 66] M.J. Flynn. Very High Speed Computer Systems. Proc. of IEEE, Vol. 54, 1966, pp. 1901-1909.

[Fulle 78] S.H. Fuller, et al. Multi-Microprocessors: An Overview and Working Example. Proceedings of the IEEE, Feb. 1978, pp. 216-228.

[Gilbe 82] R. Gilbert. The General Purpose Interface Bus. IEEE Micro, Vol. 2, No. 1, Feb. 1982, pp. 41-51.

[Gordo 75] G.Gordon. The Application of GPSS V to Discrete System Performance. ACM Computing Surveys, Vol. 10, No. 3, 1978, pp. 219-224.

[Gottl 82] A. Gottlieb, and J.T. Schwartz. Networks and Algorithms for Very Large Scale Parallel Computation. IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 27-36.

[Graha 78] G.S. Graham. Queuing Network Models of Computer System Performance. ACM Computing Surveys, Vol. 10, No. 3, 1978, pp. 219-224.

[Hanse 81] B. Hansen. Edison - A Multiprocessor Language. Software Practice and Experience, Vol. 11, 1981, pp.325-361.

[Hayne 82] L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell. A Survey of Highly Parallel Computing. IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 9-24.

[Hirsc 79] A.D. Hirschman, R. Swan, and G. Ali. Standard Modules Offer Flexible Multiprocessor System Design. Computer Design, May 1979, pp. 181-189.

[Intel 80] ------ Intel 2920 Analog Signal Processor Handbook. Intel Corporation, Aug. 1980.

[Jones 79] A.K. Jones, et al. Star-OS, A Multiprocessor Operating System for the Support of Task Forces. Proc. 7th Symp. Operating Systems Principles, SIGOPS, 1979, pp. 117-127.

[Jones 80] A.K. Jones, and P. Schwarz. Experience Using Multiprocessor Systems - A Status Report. ACM Computing Surveys, Vol. 12, No. 2, June 1980.

[Karta 79] S.I. Kartashev, S.P. Kartashev, and C.V. Ramamoorthy. Adaptation Properties for Dynamic Architectures. AFIPS Conference Proceedings, Vol. 48, 1979, pp. 543-556.

[Karta 82] S.P. Kartashev. Supersystems: Current State of the Art Guest Editor's Introduction. IEEE

Trans. on Computers. Vol. C-31, No. 5, May 1982.

[Klee 82] K. Klee, J.W. Verity, and J. Johnson. Battle of the Networks. Datamation, March 1982, pp. 114-117.

[Klein 75] L.Kleinrock. Queueing Systems I. John Wiley, New York, 1975.

[Kober 77] R. Kober. The Multiprocessor System SMS 201 Combining 128 Microprocessors to a Powerful Computer. COMPCON Fall 1977, pp. 225-230.

[Kogge 80] P.M. Kogge. The Architecture of Pipelined Computers. McGraw-Hill, 1980.

[Kuck 77] D.J. Kuck. A Survey of Parallel Machine Organization and Programming. Computing Surveys, Vol. 9, No. 1, March 1977, pp. 29-60.

[Kuck 82] D.J. Kuck, and R.A. Stokes. The Burroughs Scientific Processor (BSP). IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 363-376.

[Kung 82] H.T. Kung. Why Systolic Architectures. IEEE Computer, Vol. 15, No. 1, Jan. 1982, pp. 37-46.

[Leben 82] J. Lebensold, T. Radhakrishnan, and W.M. Jaworski. A Modelling Tool for Office Information Systems. Proc of SIGOA Conf. on Office Information Systems. June 1982, pp.141-153.

[Lecou 81] M.P. Lecouffe. A Multiprocessor Architecture Using a Circulating Memory. Trends in Information Processing Systems, 3rd Conference of the European Cooperation in Informatics, Munich, Oct. 1981.

[Linco 82] N.R. Lincoln. Technology and Design Tradeoffs in the Creation of a Modern Supercomputer. IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 349-362.

[Makou 75] J. Makoul. Linear Prediction. A Tutorial Review. Proc. IEEE, Vol. 63, 1975, pp. 561-580.

[Mead 80] C. Mead, and L. Conway. Introduction to VLSI Systems. Addison Wesley, Reading, Mass., 1980.

[Metca 76] R.M. Metcalf, and D.R. Boggs. ETHERNET: Distributed Packet Switching for Local Computer Networks. Comm. of the ACM, Vol. 19, No. 7, July 1976.

[Murth 80] S.M. Murtha and M. Waite. CP/M Primer. Howard W. Sams Co., 1980.

[Oskar 77] E.A. Oskarahan, and K.C. Sevcik. Analysis of Architectural Features for Enhancing the Performance of a Database Machine. ACM Trans. on Database Systems, Vol. 2, No. 4, Dec. 1977, pp. 297-316.

[Santo 81] L.F. Santora. IEEE 488 Error Handling Techniques: Pros and Cons. Computer Design, June 1981, pp. 143-147.

[Seeth 82] S. Seetharaman, T. Radhakrishnan, and C.Y. Suen. Real Time Linear Predictave Analysis of Speech Using Multimicrocomputers. Midwest Symposium on Circuits and Systems, Sept 1982.

[Siege 79] H.J. Siegel, R.J. McMillan, and P.T. Mudler. A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems. AFIPS Conference Proceedings, Vol. 48, 1979, pp. 529-542.

[Siewi 82] D.P. Siewiorek, C.G. Bell, and A. Newell. Computer Structures: Principles and Examples. McGraw-Hill, 1982.

[Silve 82] G. Silverman, A. Stundel, and J. Lehman. A Model for Laboratory Instrument Design. IEEE Micro, Vol. 2, No. 2, May 1982, pp. 51-62.

[Snyde 82] L. Snyder. Introduction to the Configurable Highly Parallel Computer. IEEE Computer. Vol. 15, No. 1, Jan. 1982, pp. 47-56.

[Stone 75] H.S. Stone. Introduction to Computer Architecture. Science Research Associates Inc., 1975.

[Swan 77] R.J. Swan, S.H. Fuller, and D.P. Siewiorek. Cm* - A Modular Multimicroprocessor. Proc. AFIPS Nat. Comp. Conf., 1977, pp. 637-644.

[Swart 82] E.E. Swartzlander, and B.K. Gilbert. Supersystems: Technology and Architecture. IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 399-409.

[Tanen 81] A.S. Tannenbaum. Computer Networks. Prentice Hall, 1981, p287.

[Techn 82] ------ Business Communications - Local Area Networks Provide an Architecture for the Expanded Use of Automated Office Equiptment. IEEE Spectrum, Vol. 19, No. 1, Jan. 1982.

[Thurb1 79] K.J. Thurber. Parallel Processor Architectures Part 1: General Purpose Systems. Computer Design, Jan. 1979, pp. 89-97.

[Thurb2 79] K.J. Thurber. Parallel Processor Architectures Part 2: Special Purpose Systems. Computer Design, Feb. 1979, pp. 103-114.

[Ullma 80] J.D. Ullman. Principles of Database Systems. Computer Science Press, 1980.

[Venka 77] K. Venkatesh. A Microprocessor Based Character Recognition System. Master's Thesis, Concordia University, 1977.

[Vick 80] C.R. Vick, S.P. Kartashev, and S.I. Kartashev. Adaptable Architectures. IEEE Computer, Nov. 1980, pp. 17-35.

[Weitz 80] C. Weitzman. Distributed Micro/Minicomputer Systems. Prentice-Hall, 1980.

[Willi 79] R.M. Williams. LSI Chips Ease Standard 488 Bus Interfacing. Computer Design, Oct. 1979, pp. 123-131.

[Witti 78] L.D. Wittie. MICRONET: A Reconfigurable Microcomputer Network for Distributed Systems Research. Simulation, Sept. 1978.

[Witti 80] L.D. Wittie, and A.M. Vantilborg. Micros, A Distributed Operating System for Micronet: A Reconfigurable Network Computer. IEEE Trans. on Computers, Vol. C-29, No. 12, Dec. 1980.

[Wolf 74] W.A. Wolf, R. Levin, and C. Pierson. Hydra: The Kernel of a Multiprocessor Operating System. Comm. ACM, Vol. 17, No. 6, 1974, pp. 337-345.

[Wulf 72] W.A. Wulf, and C.G. Bell. C.mmp; A Multi-Miniprocessor. AFIPS Fall Jt. Computer Conf., Dec. 1972.

APPENDIX I

```
ITV. GPSS V/6000                    CRM GPSS V/6000 VER. 2.0

 *LOC   OPERATION  A,B,C,D,E,F,G,H,I,J        COMMENTS

 *
 * SIMULATION OF COMPUTER BUS - SYSTEM ONE
 * MESSAGE SIZE MEAN 15 DEV=5, ARRIVAL TIME FOR TYPE A MESSAGE= 100
 * SHARED MEMORY BURST 20% OFF 80%
 *
  SIMULATE
 NORM FUNCTION RN1,C25
 0,-5/.00003,-4/.00135,-3/.00621,-2.5/.02275,-2/.06681,-1.5
 .11507,-1.2/.15866,-1/.21186,-.8/.27425,-.6/.34458,-.4/.42074,-.2
 .5,0/.57926,.2/.65542,.4/.72575,.6/.78814,.8/.84134,1/.88493,1.2
 .93319,1.5/.97725,2/.99379,2.5/.99865,3/.99997,4/1,5
 *
 EXP FUNCTION RN1,C24   EXPONENTIAL DISTRIBUTION MEAN 1
 0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69
 .6,.915/.7,1.2/.75,1.38/.8,1.6/.84,1.83/.88,2.12
 .9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/.97,3.5
 .98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
 *
 DEC1 FUNCTION RN1,D2
 .5,5/1,6
 *
 DEC2 FUNCTION RN1,D2
 .5,4/1,6
 *
 DEC3 FUNCTION RN1,D2
 .5,4/1,5
 *
 DEC4 FUNCTION RN1,D2
 .5,1/1,2
 *
 DEC5 FUNCTION RN1,D3
 .333,4/.667,5/1,6
 *
 DEC6 FUNCTION RN1,D2
 .5,8/1,9
 *
 SNDQ FUNCTION PH3,S13,Q
 1,SMQ1/2,SMQ2/3,SMQ3/4,SMQ4/5,SMQ5/6,SMQ6
 7,SMQ7/8,SMQ8/9,SMQ9/10,SMQ10/11,SMQ11/12,SMQ12
 13,SMQ13
 *
 RSCQ FUNCTION PH1,S10,Q
 1,RCS1/2,RCS2/3,RCS3/4,RCS4/5,RCS5/6,RCS6
 7,RCS7/8,RCS8/9,RCS9/10,RCS10
 *
 * GENERATE THE REQUESTS FROM THE SHARED MEMEORY UNITS
 *
 CMU11 GENERATE ,,0,1  GENERATE CONTROL TRANSACTION FOR SHARED MEMORY
 HIGH1 LOGIC S MEM11   TURN ON MEMORY UNIT ONE
  ADVANCE 12600,0      WAIT THE TIME OF A BURST OF TRANSFERS
  LOGIC R MEM11        TURN OFF THE BURST OF MEMORY TRANSFERS
  ADVANCE V$TBMT       WAIT TIME BETWEEN MEMORY TRANSFERS
  TRANSFER ,HIG-11
 *
 MMU11 GENERATE 700,FNSEXP GENERATE SHARED MEMORY MESSAGES FOR UNIT
```

```
ITY. GPSS V/6000                          CRM GPSS V/6000 VER. 2.0

 *LOC    OPERATION  A,B,C,D,E,F,G,H,I,J        COMMENTS

  GATE LS MEM11 TEST IF UNIT BURST IS REQUIRED
  ASSIGN 1,FN$DEC4,PH ASSIGN DESTINATION
  ASSIGN 2,V$LEN2,PH ASSIGN MESSAGE LENGTH
  ASSIGN 3,11,PH ASSIGN SENDING QUEUE
  QUEUE SMQ11
  TRANSFER ,GTBUS
 *
 CMU12 GENERATE ,,0,1 CONTROL TRANSACTION FOR SHARED MEMORY UNIT TWO
 HIGH2 LOGIC S MEM12
  ADVANCE 12600,0
  LOGIC R MEM12
  ADVANCE V$TBMT
  TRANSFER ,HIGH2
 *
 MMU12 GENERATE 700,FN$EXP GENERATE SHARED MEMORY TRANSFERS FOR UNIT
  GATE LS MEM12
  ASSIGN 1,FN$DEC5,PH
  ASSIGN 2,V$LEN2,PH
  ASSIGN 3,12,PH
  QUEUE SMQ12
  TRANSFER ,GTBUS
 *
 CMU13 GENERATE ,,0,1 CONTROL TRANSACTION FOR SHARED MEMORY UNIT THR
 HIGH3 LOGIC S MEM13
  ADVANCE 12600,0
  LOGIC R MEM13
  ADVANCE V$TBMT
  TRANSFER ,HIGH3
 *
 MMU13 GENERATE 700,FN$EXP GENERATE SHARED MEMORY MESSAGES FROM UNIT
  GATE LS MEM13
  ASSIGN 1,FN$DEC6,PH
  ASSIGN 2,V$LEN2,PH
  ASSIGN 3,13,PH
  QUEUE SMQ13
  TRANSFER ,GTBUS
 *
 * GENERATE THE TRANSFER REQUESTS FROM THE SLAVE UNITS
 *
 SMG1 GENERATE 1000000,FN$EXP  GENERATE THE MESSAGES FROM SLAVE ONE
  ASSIGN 1,2,PH  ASSIGN DESTINATION SLAVE
  ASSIGN 2,V$LEN1,PH ASSIGN MESSAGE LENGTH
  ASSIGN 3,1,PH ASSIGN SEND SLAVE QUEUE NUMBER
  QUEUE SMQ1
  TRANSFER ,GTBUS
 *
 SMG2 GENERATE 1000000,FN$EXP      MESSAGES SENT BY SLAVE TWO
  ASSIGN 1,3,PH
  ASSIGN 2,V$LEN1,PH
  ASSIGN 3,2,PH
  QUEUE SMQ2
  TRANSFER ,GTBUS
 *
 SMG3 GENERATE 1000000,FN$EXP      MESSAGES FOR SLAVE THREE
  ASSIGN 1,10,PH
```

```
*LOC    OPERATION  A,B,C,D,E,F,G,H,I,J        COMMENTS

 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,3,PH
 QUEUE SMQ3
 TRANSFER ,GTBUS
*
SMG4 GENERATE 1000000,FN$EXP     MESSAGES FOR SLAVE FOUR
 ASSIGN 1,1,FN$DEC1,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,4,PH
 QUEUE SMQ4
 TRANSFER ,GTBUS
*
SMG5 GENERATE 1000000,FN$EXP MESSAGES FOR SLAVE FIVE
 ASSIGN 1,1,FN$DEC2,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,5,PH
 QUEUE SMQ5
 TRANSFER ,GTBUS
*
SMG6 GENERATE 1000000,FN$EXP  MESSAGES FOR SLAVE SIX
 ASSIGN 1,1,FN$DEC3,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,6,PH
 QUEUE SMQ6
 TRANSFER ,GTBUS
*
SMG7 GENERATE 1000000,FN$EXP MESSAGES FOR SLAVE 7
 ASSIGN 1,8,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,7,PH
 QUEUE SMQ7
 TRANSFER ,GTBUS
*
SMG8 GENERATE 1000000,FN$EXP MESSAGES FOR SLAVE EIGHT
 ASSIGN 1,9,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,9,PH
 QUEUE SMQ9
 TRANSFER ,GTBUS
*
SMG9 GENERATE 1000000,FN$EXP MESSAGES FOR SLAVE NINE
 ASSIGN 1,8,PH
 ASSIGN 2,V$LEN1,PH
 ASSIGN 3,9,PH
 QUEUE SMQ9
 TRANSFER ,GTBUS
*
*   PERFORM THE BUS CONTROLLER FUNCTIONS
*
GTBUS ENTER BUS
 DEPART FN$SNDU    MARK THE MESSAGE AS LEAVING THE SLAVE BUFFER
 ADVANCE V$BTIME   WAIT FOR THE SERVICE TIME OF THE CONTROLLER
 LEAVE BUS    FREE THE BUS CONTROLLER
*
* TRANSFER TO DESTINATION QUEUE
```

```
TY. GPSS V/6000                          CRM GPSS V/6000 VER. 2.0

 *LOC   OPERATION  A,B,C,D,E,F,G,H,I,J        COMMENTS

 *
  QUEUE FN$RSCQ        ADD TRANSACTION TO THE RECIEVE QUEUE
  ADVANCE V$STIME      WAIT THE TIME THE SLAVE NEEDS TO CLEAR THE BUFF
  DEPART FN$RSCQ LEAVE THE OUTPUT QUEUE
  TERMINATE 1       THE TRANSACTION LEAVES THE SYSTEM
 *
 * DECLARE VARIABLES
 *
 LEN1 VARIABLE 5*FN$NORM+15
 LEN2 VARIABLE 10*FN$NORM+64
 TBMT VARIABLE 50+00*FN$EXP
 BTIME VARIABLE (PH2+7)4+100
 STIME VARIABLE (PH2+7)*8
  STORAGE S$BUS,1
 *
 *CNTRL OF SIMULATION
 *
  START  30,NP
```

APPENDIX II

```
                MAN   DBUG
00F0            SETDP $F0
        *
        * THIS ROUTINE WILL DRIVE THE C-BUS
        * CONTROLLER  WHICH IS BASED ON
        * A 6809 MICROPROCESSOR
        *


        *
        * SYSTEM EQUATES
        *
FB10    PDATA1  EQU   $FB10
FB0C    PSTRNG  EQU   $FB0C
FB06    INCHE   EQU   $FB06
FB08    INCHEK  EQU   $FB08
FB0E    PCRLF   EQU   $FB0E
FB0A    OUTCH   EQU   $FB0A
CD03    WARMS   EQU   $CD03
        *
        * BUS CONTROLLER HARDWARE ADDRESSES
        *
F000    SADDU   EQU   $F000     SEND ADD UPPER BYTE
F001    SADDL   EQU   $F001     SEND ADD LOWER BYTE

F002    RADDU   EQU   $F002     RECIEVE ADD UPPER BYTE
F003    RADDL   EQU   $F003     RECVIEVE ADD LOWER BYTE
F004    RADINC  EQU   $F004     RECIEVE ADD INCREMENT
F005    SADINC  EQU   $F005     SEND ADD INCREMENT
F006    PARITY  EQU   $F006     PARITY STATUS BIT
F007    BUSREQ  EQU   $F007     BUS REQUEST STATUS
F008    BSGNTO  EQU   $F008     BUS GRANT ISSUE
F009    PRSET   EQU   $F009     PARITY STATUS RESET
F00A    BSGNTF  EQU   $F00A     BUS GRANT OFF
F00B    MADD    EQU   $F00B     MASTER BASE ADDRESS
F010    BYTESND EQU   $F010     SEND BYTE BETWEEN PROCESSORS
F011    RDBYTE  EQU   $F011     READ BYTE FROM PROCESSOR
F012    WRTBYTE EQU   $F012     WRITE BYTE TO PROCESSOR
F00C    RDADD   EQU   $F00C     READ C-BUS DATA LINES
F00D    BGNTAK  EQU   $F00D     BUS GRANT ACKNOWLEDGE STATUS BIT
F00E    SLACK   EQU   $F00E     SLAVE ACKNOWLEDGE STATUS BIT
        *
        * INTERFACE MASKS USED TO CALCULATE
        * A BASE ADDRESS
        *
0000    OUTPTM  EQU   $00       OUTPUT BUFFER
0000    INPUTM  EQU   $00       INPUT BUFFER
0001    ACCSM   EQU   $01       ACCESS VECTOR

0001    SNDCTM  EQU   $01       SEND INTERFACE CONTROL
```

```
0001 RCVCTN EQU   $01      RECIEVE INTERFACE CONTROL
     *
     * LEAST SIGNIFICANT BYTE OF ADDRESS FOR INTERFACE
     * HARDWARE CONTROL STRUCTURES
     *
0000 OUTLA  EQU   $00      OUTPUT BUFFER
0000 INPTLA EQU   $00      INPUT BUFFER
0000 ACCSLA EQU   $00      ACCES VECTOR
0000 FULRST EQU   $00      FULL BIT RESET
0001 MKBTST EQU   $01      MASK BIT SET
0002 MKBTRST EQU  $02      MASK BIT RESET
0080 STINBT EQU   $80      SET INPUT BIT
0003 RDINBT EQU   $03      READ INPUT BIT
     *
     * SYSTEM BUFFER
     *
0000         STACK  RMB   30
001E         STCXPT RMB   2
0020         STCOND RMB   1
0021         TEMPW  RMB   2
0023         LCOUNT RMB   1
0024         CSFLG  RMB   1

0025         WLCOUNT RMB  1
     *
     * MAIN COMMAND LOOP
     *
0026 8E 04E5 DEBUG LDX    #STRT
0029 AD 9F F810    JSR    [PDATA1]
002D 96 09         LDA    PRSET     RESET PARITY
002F 8E 0000       LDX    #STACK
0032 BF 001E       STX    STCXPT    SET STACK POINTER
0035 7F 0020       CLR    STCOND
0038 86 F0         LDA    #$F0
003A 1F 8B         TFR    A,DP      SET DP REG
003C 8E 04F1 CMTCP LDX    #PRMPT
003F AD 9F F810    JSR    [PDATA1]  GIVE PROMPT
0043 AD 9F F806    JSR    [INCHE]
0047 81 47         CMPA   #'G
0049 1027 011A     LBEQ   GOO
004D 81 49         CMPA   #'I
>004F 1027 002F    LBEQ   INPTR
0053 81 4F         CMPA   #'O
>0055 1027 0049    LBEQ   OUTPT
0059 81 41         CMPA   #'A
005D 1027 00BC     LBEQ   ACCES

005F 81 53         CMPA   #'S
0061 1027 00A9     LBEQ   SENDP
0065 81 43         CMPA   #'C
0067 1027 00F6     LBEQ   CONTM
006B 81 52         CMPA   #'R
006D 1027 00C5     LBEQ   RECV
0071 81 51         CMPA   #'Q
```

```
0073 26  03                BNE   INCH
0075 7E  C003              JMP   WARMS
0078 8E  050A     INCH     LDX   #INUCH
007B AD  9F F810           JSR   [PDATA1]
007F 20  BB                BRA   CMDLP
                  *
                  * FILL THE INPUT BUFFER OF THE GIVEN INTERFACE
                  *
0081 8E  0419     INPTB    LDX   #SPACE
0084 AD  9F FB0C           JSR   [PSTRNG]
0088 BD  0338              JSR   INHEX
>008B 1025 FFE9            LBCS  INCH
008F 8A  00                ORA   #INPUTH
0091 97  02                STA   RADDU
0093 86  00                LDA   #INPTLA   SET THE C-BUS
0095 97  03                STA   RADDL     ADDRESS

0097 4F                    CLRA
0098 97  12       LOOP1    STA   WRTBYTE
009A B6  04                LDB   RADINC    SEND PATTERN
009C 4C                    INCA
009D 26  F9                BNE   LOOP1
>009F 7E  003C             JMP   CMDLP
                  *
                  * READ THE OUTPUT BUFFER
                  *
00A2 8E  0419     OUTPT    LDX   #SPACE
00A5 AD  9F FB0C           JSR   [PSTRNG]
00A9 BD  0338              JSR   INHEX
>00AC 1025 FFC8            LBCS  INCH
00B0 8A  00                ORA   #OUTPTH
00B2 97  00                STA   SADDU     SET THE SEND ADDRESS
00B4 86  00                LDA   #OUTLA
00B6 97  01                STA   SADDL
00B8 AD  9F FB0E           JSR   [PCRLF]
00BC 7F  0025              CLR   BLCOUNT
00BF 7F  0023     OUTER    CLR   LCOUNT
00C2 AD  9F FB0E           JSR   [PCRLF]
00C6 96  11       INNER    LDA   RDBYTE    GET NEXT BYTE
00C8 BD  0352              JSR   OUT2HS    TO TERMINAL

00CB 8E  0419              LDX   #SPACE
00CE AD  9F FB0C           JSR   [PSTRNG]
00D2 96  05                LDA   SADINC    INC SEND ADD. C-BUS
00D4 7C  0023              INC   LCOUNT
00D7 B6  0023              LDA   LCOUNT
00DA 81  10                CMPA  #$10      END OF LINE
00DC 26  E8                BNE   INNER
00DE 7C  0025              INC   BLCOUNT
00E1 B6  0025              LDA   BLCOUNT
00E4 81  10                CMPA  #$10      END OF BLOCK
00E6 26  D7                BNE   OUTER
00E8 7E  003C             JMP   CMDLP
                  *
```

```
                    * LOAD THE ACCESS  VECTOR WITH TEST PATTEN
                    *
00EB BE   0419      ACCES  LDX   #SPACE
00EE AD   9F F80C          JSR   [PSTRNG]
00F2 BD   0338             JSR   INHEX
00F5 1025 FF43             LBCS  CMDLP
00F9 8A   01               ORA   #ACCSH
00FB 97   02               STA   RADDU     SET THE RECEIVE ADDRESS
00FD 86   00               LDA   #ACCSLA
00FF 97   03               STA   RADDL

0101 4F                    CLRA
0102 97   12        LOOPA  STA   WRTBYTE
0104 B6   04               LDA   RADINC
0106 4C                    INCA
0107 81   10               CMPA  #$10      SEND PATTERN
0109 20   F7               BLT   LOOPA
010B 7E   003C             JMP   CMDLP
                    *
                    *
                    * SENDP TO THE GIVEN ADDRESS "55"
                    *
010E BE   0419      SENDP  LDX   #SPACE
0111 AD   9F F80C          JSR   [PSTRNG]
0115 BD   0338             JSR   INHEX
0118 1025 FF20             LBCS  CMDLP
011C 97   02               STA   RADDU
011E BD   0338             JSR   INHEX     SET THE ADDRESS
0121 97   03               STA   RADDL
0123 86   55               LDA   #$55
0125 AD   9F F808   LOOP2  JSR   [INCHEK]
0129 27   07               BEQ   OK2
012B AD   9F F806          JSR   [INCHE]
012F 7E   003C             JMP   CMDLP

0132 97   12        OK2    STA   WRTBYTE
0134 20   EF               BRA   LOOP2
                    *
                    * RECEIVE FROM THE GIVEN ADDRESS
                    *
0136 BE   0419      RECV   LDX   #SPACE
0139 AD   9F F80C          JSR   [PSTRNG]
013D BD   0338             JSR   INHEX
0140 1025 FEF8             LBCS  CMDLP
0144 97   00               STA   SADDU     SET ADDESS
0146 BD   0338             JSR   INHEX
0149 97   01               STA   SADDL
014B AD   9F F808   LOOP3  JSR   [INCHEK]
014F 27   07               BEQ   OK1
0151 AD   9F F806          JSR   [INCHE]
0155 7E   003C             JMP   CMDLP
0158 96   11        OK1    LDA   RDBYTE
015A 20   EF               BRA   LOOP3
                    *
```

```
                  * DATA BUFFERS
                  *
015C              SEND    RMB   1
015D              RECIEVE RMB   1

015E              LENGTH  RMB   1
015F              MTYPE   RMB   1
0160              HOPE    RMB   1
                  * OBTAIN ADDRESS OF SENDING PROCESSOR
                  *
0161 7F   0024    CONTM   CLR   CSFLG
0164 7E   016C            JMP   GO
0167 86   55      GOO     LDA   #$55
0169 B7   0024            STA   CSFLG
016C AD   9F F808 GO      JSR   [INCHEK]
0170 27   07              BEQ   OK
0172 AD   9F F806         JSR   [INCHE]
0176 7E   003C            JMP   CMDLP
0179 AD   9F F80E OK      JSR   [PCRLF]
017D BE   0469            LDX   #MES
0180 AD   9F F80C         JSR   [PSTRNG]
0184 96   07              LDA   BUSREQ
0186 B7   0160            STA   HOPE
0189 BD   0352            JSR   OUT2HS
018C B6   0160            LDA   HOPE
018F 2A   0B              BPL   CONN1
0191 B6   0020            LDA   STCOND
0194 27   D6              BEQ   GO        EMPTY STACK

0196 BD   0397            JSR   PULL      OF WAITING MESS.
0199 7E   0308            JMP   DMGO
019C BE   049C    CONN1   LDX   #PRO
019F AD   9F F810         JSR   [PDATA1]
01A3 96   08              LDA   BSGNTO    BUS GRANT ISSUE
01A5 5F                   CLRB
01A6 BE   04D4    BUSGRB  LDX   #BSRQGVN
01A9 AD   9F F810         JSR   [PDATA1]
01AD 5C                   INCB
01AE 96   0D              LDA   BGNTAK    WAIT FOR BUS
01B0 2A   0E              BPL   CONTG     GRANT ACKNOWLEDGE
01B2 C1   04              CMPB  #$04
01B4 2D   F0              BLT   BUSGRB
01B6 BE   041B            LDX   #NBGACK
01B9 AD   9F F810         JSR   [PDATA1]
01BD 7E   016C            JMP   GO
01C0 96   0C      CONTG   LDA   RDADD
01C2 B7   015C            STA   SEND
01C5 8A   00              ORA   #OUTPTM
01C7 97   00              STA   SADDU     GET SEND ADDRESS
01C9 86   00              LDA   #OUTLA    DRIVERS
01CB 97   01              STA   SADDL
01CD 96   0A              LDA   BSGNTF    LOWER BUS GRANT
```

*

```
                  * GET MESSAGE INFORMATION
                  *
01CF 96  05              LDA   SADINC
01D1 96  11              LDA   RDBYTE   READ SELECTIVE BYTES
01D3 B7  015D            STA   RECIEVE  FROM THE MESSAGE HEADER
01D6 96  05              LDA   SADINC
01D8 96  11              LDA   RDBYTE
01DA B7  015E            STA   LENGTH
01DD 96  05              LDA   SADINC
01DF 96  11              LDA   RDBYTE
01E1 B7  015F            STA   MTYPE
01E4 AD  9F F80E         JSR   [PCRLF]
01E8 BE  0434            LDX   #MS
01EB AD  9F F80C         JSR   [PSTRNG]
01EF B6  015C            LDA   SEND
01F2 BD  0352            JSR   OUT2HS
01F5 BE  043A            LDX   #MR
01F8 AD  9F F80C         JSR   [PSTRNG]
01FC B6  015D            LDA   RECIEVE
01FF BD  0352            JSR   OUT2HS
0202 BE  0445            LDX   #ML
0205 AD  9F F80C         JSR   [PSTRNG]

0209 B6  015E            LDA   LENGTH
020C BD  0352            JSR   OUT2HS
020F BE  044F            LDX   #MM
0212 AD  9F F80C         JSR   [PSTRNG]
0216 B6  015F            LDA   MTYPE
0219 BD  0352            JSR   OUT2HS
021C AD  9F F80E         JSR   [PCRLF]
                  *
                  * CHECK IF DESTINATION INTERFACE INPUT BUFFER
                  * IS FREE
                  *
0220 B6  015D            LDA   RECIEVE
0223 BA  01              ORA   #SNDCTH  SET SEND ADDRESS
0225 97  00              STA   SADDU    FOR INT BIT READ
0227 5F                  CLRB
0228 96  0E       SLWAIT LDA   SLACK
022A 2A  0F              BPL   CCN
022C 5C                  INCB
022D C1  04              CMPB  #$04
022F 2D  F7              BLT   SLWAIT   TIME OUT?
0231 BE  04C0            LDX   #WSL
0234 AD  9F F810         JSR   [PDATA1]
0238 7E  02FB            JMP   AENDC

023B B6  03       CCN    LDA   #RDINBT
023D 97  01              STA   SADDL
023F 96  11              LDA   RDBYTE
0241 2A  0D              BPL   ISFRF
```

```
0243 BE   048A          LDX   #DESTB
0246 AD   9F F810       JSR   [PDATA1]
024A BD   0372          JSR   PUSH
024D 7E   0308          JMP   DMCO
                *
                * CHECK MESSAGE TYPE
                *
0250 BE   04AE   ISFRE  LDX   #FRR
0253 AD   9F F810       JSR   [PDATA1]
0257 B6   015F          LDA   MTYPE
025A B5   80            BITA  #$80
025C 27   02            BEQ   NN1      NOT TYPE ZERO
025E 20   14            BRA   TRANS
0260 B6   015F   NN1    LDA   MTYPE
0263 B5   40            BITA  #$40
0265 27   03            BEQ   RPTER
0267 7E   02AD          JMP   ACCESC   TYPE TWO
026A BE   04E5   RPTER  LDX   #INTYP
026D AD   9F F810       JSR   [PDATA1]

0271 7E   02FB          JMP   AENDC
                *
                * TRANSMITT A MESSAGE WHERE THE SEND AND
                * RECIEVE ADDRESSES ARE FOUND IN THE SEND
                * AND RECIEVE POINTERS
                *
0274 F6   015C   TRANS  LDB   SEND
0277 BE   047F          LDX   #TRNS
027A AD   9F F810       JSR   [PDATA1]
027E CA   00            ORB   #OUTPTH  SET SEND ADDRESS
0280 D7   00            STB   SADDU
0282 B6   00            LDA   #OUTLA
0284 97   01            STA   SADDL
0286 B6   015D          LDA   RECIEVE
0289 8A   00            ORA   #INPUTH  SET RECIEVE ADDRESS
028B 97   02            STA   RADDU
028D 86   00            LDA   #INPTLA
028F 97   03            STA   RADDL
0291 F6   015E          LDB   LENGTH   GET NUMBER OF BYTES
0294 CB   07            ADDB  #$07     MESSAGE HEADER
0296 96   10     LOOP   LDA   BYTESND
0298 5A                 DECB           TRANSFER MESSAGE
0299 26   FB            BNE   LOOP

029B 96   10            LDA   BYTESND  SEND CHKSUM
029D 96   06            LDA   PARITY
029F 2A   4D            BPL   TENDC    NO ERROR
02A1 96   09            LDA   PRSET    RESET PARITY FLAG
02A3 BE   045C          LDX   #PERR
02A6 AD   9F F810       JSR   [PDATA1]
02AA 7E   02EE          JMP   TENDC
```

```
                *
                * SET THE ACCESS VECTOR OF THE
                * DESTINATION PROCESSOR
                *
02AB 96   0B    ACCESC LDA   MADD
02AF B1   015C         CMPA  SEND     IS THE MASTER MAKING THE
02B2 27   0A           BEQ   CONT1    REQUEST
02B4 BE   0470         LDX   #INVACC
02B7 AD   9F F810      JSR   [PDATA1] ISSUE ERROR MESSAGE
>02BB 7E  02FB         JMP   AENDC
02BE B6   015C  CONT1  LDA   SEND
02C1 8A   00           ORA   #OUTPTH
02C3 97   00           STA   SADDU
02C5 86   07           LDA   #7
02C7 8B   00           ADDA  #OUTLA   SET THE SEND ADDRESS
02C9 97   01           STA   SADDL

02CB B6   015D         LDA   RECIEVE
02CE 8A   01           ORA   #ACCSH
02D0 97   02           STA   RADDU    SET THE RECIEVE
02D2 86   00           LDA   #ACCSLA  ADDRESS
02D4 97   03           STA   RADDL
02D6 F6   015E         LDB   LENGTH
02D9 96   10    LOOP1A LDA   BYTESND
02DB 5A                DECB
02DC 26   FB           BNE   LOOP1A
02DE 96   06           LDA   PARITY   CHECK FOR PARITY
02E0 2A   19           BPL   AENDC
02E2 96   09           LDA   PRSET
02E4 BE   045C         LDX   #PERR    ISSUE MESSAGE
02E7 AD   9F F810      JSR   [PDATA1]
>02EB 7E  02FB         JMP   AENDC
                *
                * MARK MESSAGE AS SENT AND RELEASE THE INTERFACES
                *
02EE B6   015D  TENDC  LDA   RECIEVE
02F1 8A   01           ORA   #RCVCTH
02F3 97   02           STA   RADDU
02F5 86   80           LDA   #STINBT  SET THE INT BIT
02F7 97   03           STA   RADDL

02F9 96   12           LDA   WRTBYTE
02FB B6   015C  AENDC  LDA   SEND
02FE 8A   01           ORA   #SNDCTH
0300 97   00           STA   SADDU    RESET THE FULL BIT
0302 86   00           LDA   #FULRST
0304 97   01           STA   SADDL
0306 96   11           LDA   RDBYTE
0308 B6   0024  DMCO   LDA   CSFLG
030B 1026 FD2D         LBNE  CMDLP
030F 7E   016C         JMP   GO
```

```
                 *
                 * GET HEX DIGIT FROM SCREEN
                 *
0312 AD  9F F806 GETHEX JSR   [INCHE]
0316 80  30              SUBA  #$30
0318 2B  19              BMI   ER1
031A 81  09              CMPA  #$09
031C 2F  0A              BLE   IN1HG
031E 81  11              CMPA  #$11
0320 2D  11              BLT   ER1
0322 81  16              CMPA  #$16
0324 2E  0D              BGT   ER1
0326 80  07              SUBA  #7

0328 1F  89      IN1HG   TFR   A,B
032A 4F                  CLRA
032B 1F  01              TFR   B,X
032D 5F                  CLRB
032E 5C                  INCB
032F 86  00              LDA   #$00
0331 47                  ASRA
0332 39                  RTS
0333 5F          ER1     CLRB          SET PARAM FOR ERROR
0334 86  01              LDA   #$01
0336 47                  ASRA
0337 39                  RTS
                 *
                 * INPUT A HEX BYTE FROM THE
                 * TERMINAL RESULT IN A-REG IN BINARY
                 *
>0338 BD 0312    INHEX   JSR   GETHEX
0338 25  14              BCS   STP
0330 1F  10              TFR   X,D
033F 86  10              LDA   #16
0341 3D                  MUL
0342 FD  0021            STD   TEMPW
>0345 BD 0312            JSR   GETHEX

0348 25  07              BCS   STP
034A 1F  10              TFR   X,D
034C F3  0021            ADDD  TEMPW
034F 1F  98              TFR   B,A
0351 39          STP     RTS
                 *
                 * OUTPUT A BYTE IN HEX FORM
                 *
0352 B7  0021    OUT2HS  STA   TEMPW
>0355 BD 035F            JSR   OUTHL   DO LEFT
0358 B6  0021            LDA   TEMPW
>0359 BD 0363            JSR   OUTHR   DO RIGHT
035E 39                  RTS
```

```
035F 44          OUTHL  LSRA
0360 44                 LSRA
0361 44                 LSRA
0362 44                 LSRA
0363 84  0F      OUTHR  ANDA  #$0F
0365 8B  30             ADDA  #$30     CONVERT TO BINARY
0367 81  39             CMPA  #$39     ASCII
0369 23  02             BLS   C01
036B 8B  07             ADDA  #$7
036D AD  9F F80A C01    JSR   [OUTCH]

0371 39                 RTS
                 *
                 * ADD THE CURRENT SENDER TO THE STACK
                 * AND SET HIS MASK BIT
                 *
0372 B6  015C    PUSH   LDA   SEND      STORE THE PROCESSORS
0375 BE  001E           LDX   STCKPT    ADRESS
0378 A7  84             STA   0,X
037A 30  01             LEAX  +1,X
037C BF  001E           STX   STCKPT    UPDATE STACK
037F 8A  01             ORA   #SNDCTM   POINTER
0381 97  00             STA   SADDU
0383 86  01             LDA   #MKBTST   SET MASK BIT
0385 97  01             STA   SADDL
0387 96  11             LDA   RDBYTE
0389 7C  0020           INC   STCOND
038C 8E  03DF           LDX   #SPMESS
038F AD  9F F810        JSR   [PDATA1]
0393 BD  03BD           JSR   SPRINT
0396 39                 RTS
                 *
                 * PULL ALL ADDRESSES FROM THE STACK
                 * AND RESET THEIR FULL BITS

                 *
0397 7F  0020    PULL   CLR   STCOND
039A 8E  03F4           LDX   #SPLMESS
039D AD  9F F810        JSR   [PDATA1]
03A1 BE  001E           LDX   STCKPT
03A4 8C  0000    CONT5  CMPX  #STACK    STACK EMPTY?
03A7 27  13             BEQ   DNE1
03A9 A6  1F             LDA   -1,X
03AB 30  1F             LEAX  -1,X     UPDATE STACK POINTER
03AD BF  001E           STX   STCKPT
03B0 8A  01             ORA   #SNDCTM
03B2 97  00             STA   SADDU    SET UP MASK BIT
03B4 86  02             LDA   #MKBTRST RESET
03B6 97  01             STA   SADDL
03B8 96  11             LDA   RDBYTE
03BA 20  E8             BRA   CONT5
03BC 39          DNE1   RTS
                 *
```

```
                    * PRINT THE CONTENTS OF THE STACK
                    *
03B0 BE   0409      SPRINT  LDX    #STACKC
03C0 AD   9F F810           JSR    [PDATA1]
03C4 10BE 001E              LDY    STCXPT

03C8 108C 0000      AGAIN   CMPY   #STACK
03CC 27   10                BEQ    END1
03CE A6   3F                LDA    -1,Y
03D0 BD   0352              JSR    OUT2HS   VALUE
03D3 BE   0419              LDX    #SPACE
03D6 AD   9F F80C           JSR    [PSTRNG] SPACE
03DA 31   3F                LEAY   -1,Y     NEXT VALUE
03DC 20   EA                BRA    AGAIN
03DE 39             END1    RTS
                    *
                    * MESSAGES
                    *
03DF 53 54 41 43    SPMESS  FCC    "STACK PUSH OPERATION"
03E3 4B 20 50 55
03E7 53 48 20 4F
03EB 50 45 52 41
03EF 54 49 4F 4E
03F3 04                     FCB    $04
03F4 53 54 41 43    SPLMESS FCC    "STACK PULL OPERATION"
03F8 4B 20 50 55
03FC 4C 4C 20 4F
0400 50 45 52 41
0404 54 49 4F 4E

0408 04                     FCB    $04
0409 53 54 41 43    STACKC  FCC    "STACK CONTENTS "
040D 4B 20 43 4F
0411 4E 54 45 4E
0415 54 53 20
0418 04                     FCB    $04
0419 20             SPACE   FCC    " "
041A 04                     FCB    $04
041B 4E 4F 20 42    NBGACK  FCC    "NO BUS GRANT ACKNOWLEDGE"
041F 55 53 20 47
0423 52 41 4E 54
0427 20 41 43 4B
042B 4E 4F 57 4C
042F 45 44 47 45
0433 04                     FCB    $04
0434 53 45 4E 44    MS      FCC    "SEND "
0438 20
0439 04                     FCB    $04
043A 20 20 52 45    MR      FCC    "  RECEIVE "
043E 43 45 49 56
0442 45 20
0444 04                     FCB    $04
0445 20 20 4C 45    ML      FCC    "  LENGTH "

0449 4E 47 54 48
```

```
044D 20
044E 04                    FCB    $04
044F 20 20 4D 45   MM      FCC    "  MESS TYPE "
0453 53 53 20 54
0457 59 50 45 20
045B 04                    FCB    $04
045C 50 41 52 49   PERR    FCC    "PARITY ERROR"
0460 54 59 20 45
0464 52 52 4F 52
0468 04                    FCB    $04
0469 57 41 49 54   MES     FCC    "WAIT  "
046D 20 20
046F 04                    FCB    $04
0470 49 4E 56 41   INVACC  FCC    "INVALID ACCESS"
0474 4C 49 44 20
0478 41 43 43 45
047C 53 53
047E 04                    FCB    $04
047F 54 52 41 53   TRNS    FCC    "TRASN MESS"
0483 4E 20 4D 45
0487 53 53
0489 04                    FCB    $04

048A 49 4E 54 45   DESTB   FCC    "INTERFACE IS BUSY"
048E 52 46 41 43
0492 45 20 49 53
0496 20 42 55 53
049A 59
049B 04                    FCB    $04
049C 42 55 53 20   PRQ     FCC    "BUS REQUEST SEEN "
04A0 52 45 51 55
04A4 45 53 54 20
04A8 53 45 45 4E
04AC 20
04AD 04                    FCB    $04

04AE 49 4E 54 45   FRR     FCC    "INTERFACE IS FREE"
04B2 52 46 41 43
04B6 45 20 49 53
04BA 20 46 52 45
04BE 45
04BF 04                    FCB    $04
04C0 4E 4F 20 53   NSL     FCC    "NO SLAVE AT ADDRESS"
04C4 4C 41 56 45
04C8 20 41 54 20
04CC 41 44 44 52

04D0 45 53 53
04D3 04                    FCB    $04
04D4 42 55 53 20   BSRQGVN FCC    "BUS GRANT GIVEN "
04D8 47 52 41 4E
04DC 54 20 47 49
04E0 56 45 4E 20
04E4 04                    FCB    $04
```

```
04E5 43 2D 42 55   STRT   FCC   "C-BUS DEBUG"
04E9 53 20 44 45
04ED 42 55 47
04F0 04                   FCB   $04
04F1 0D0A          PRMPT  FDB   $0D0A
04F3 3E                   FCC   ">"
04F4 04                   FCB   $04
04F5 49 4E 56 41   INTYP  FCC   "INVALID MESSAGE TYPE"
04F9 4C 49 44 20
04FD 4D 45 53 53
0501 41 47 45 20
0505 54 59 50 45
0509 04                   FCB   $04
050A 0D0A          INVCM  FDB   $0D0A
050C 49 4E 56 41          FCC   "INVALID COMMAND"
0510 4C 49 44 20

0514 43 4F 4D 4D
0518 41 4E 44
051B 04                   FCB   $04
                          END   DBUG
```