AD-A115 639 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 17/2
PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFA--ETC(U)
DEC 81 A G GRAVIN

UNCLASSIFIED AFIT/GCS/EE/81D-9 NL

1 OF 2
AD A 115639

DDC

1

AD A115639

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/81D-9

PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFACE USING INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

THESIS

AFIT/GCS/EE/81D-9 Andrew G. Gravin
2Lt USAF

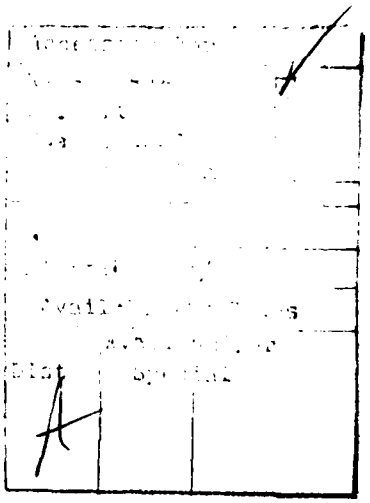
Approved for public release; distribution unlimited.

AFIT/GCS/EE/81D-9

PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFACE USING INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science



by
Andrew G. Gravin
2Lt, USAF
Graduate Computer Systems
December 1981

Preface

This research effort describes the preliminary hardware design of an improved Universal Network Interface Device (UNID II). The concept of a universal network interface was studied in previous Air Force Institute of Technology research investigations. This report further defines and clarifies the concept of a network interface and uses the state-of-the-art 8086 family of microprocessors to give UNID II increased performance capabilities.

I would like to sincerely thank several people whose expert guidance proved to be invaluable. Dr. Lamont, my thesis advisor, provided this research topic which enabled fulfillment of a long-time desire to study state-of-the-art microprocessors. His comments kept me motivated and helped discourage unproductive approaches. Major Seward's computer-networking course inspired a keen interest in the field and provided a basic understanding of how networks operate. Despite my arriving at the Air Force Institute of Technology (AFIT) a month later than the rest of my class, Dr. Hartrum, my academic advisor, helped me to get a good start on course work. Finally, I should thank all the uncounted persons whose efforts made the personal computer a reality. Even though I broke the hand I write with during the time I was preparing this paper, I found that modern word-processing equipment enabled text to be recorded easier and more efficiently than was previously possible with two good hands and pencil and paper.

Contents

List of Figures

List of Figures

List of Tables

Abstract

This research describes the design of a Universal Network Interface Device (UNID II) which is intended for use in a computer communications network. The II distinguishes UNID II from the original UNID which was also designed and developed at the Air Force Institute of Technology. UNID II's purpose is to lessen the time delays and development costs incurred by custom-designing network interfaces for each application. UNID II is a programmable interface; and although different applications require different device dependent programming, UNID II hardware remains essentially unchanged. A requirements study shows that to handle a wide variety of interfacing situations, UNID II must perform node functions which include concentration, switching and routing, front-end processing, and user-terminal interfacing. The performance of these functions relieves network hosts from communication-specific software.

The key design concept is the subdivision of UNID II into two independent subsystems which communicate through an area in shared memory. The Network Subsystem handles high-speed network links while the Local Subsystem handles network peripherals. This modular approach reduces bus contention and allows the effect of an error or change to be isolated.

For high performance, the Network Subsystem includes the Intel 8089 I/O processor; and for flexibility, the Local Subsystem uses a Multibus-compatible communications board which contains serial or parallel interfaces and may be interchanged depending on user needs.

PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFACE USING INTEL 8086 AND 8089 16-BIT MICROPROCESSORS

I. Introduction

In response to increased data communications requirements on a typical Air Force Base, Rome Air Development Center (RADC) specified a need for a small and economical network interface device to connect various base computers and terminals to a computer communications network. Several Air Force Institute of Technology (AFIT) research efforts have studied this need and produced a prototype Universal Network Interface Device (UNID) (Refs 3,6,31).

Using 16-bit microprocessor architecture, this research effort develops a new network interface design (UNID II) which is capable of performing the functions of a network node. Since it is designed for flexibility, UNID II can handle a wide variety of network interfacing applications. The first prototype of UNID II will be used as a node for AFIT's Digital Engineering Laboratory.

Motivation for Computer Networks

A computer network is an interconnected set of host computers and peripherals which communicate with each other and share resources such as software, data bases, memory space, and user terminals (Ref 9:111). By eliminating the need to duplicate facilities, computer networks can greatly reduce the cost of computing facilities. For example, instead of hard-wiring two adjacent user terminals to separate computers, a network can allow the use of one terminal with

either computer (Ref 40:137). The combined resources of a computer network give the user convenient access to more processing facilities than are available at any individual computer site. Since the failure of one remote component will not disable the entire network, a network is more reliable than a single computer. When a system's capability needs to be expanded, adding small and relatively inexpensive components to a network is more economical than replacing an entire mainframe computer with a larger mainframe (Ref 11:108). Some popular uses of a computer networks are shown in Table I.

Network Topologies and Node Functions

This section provides introductory information for Chapter II, Requirements Analysis. Since UNID II is designed to act as a network node capable of handling many different topologies and functions, these attributes are briefly described.

The topology discussions demonstrate environments which use a network node and outline the general requirements imposed by different topologies. For example, some topologies may require higher nodal transmission rates or more communication links per node than other topologies. Depending on the topology, failure of a single node could disable the entire network or only part of the network. (Although a network node is sometimes considered to consist of both a network communications interface and a host

Table I

Typical Uses of Computer Networks (Ref 8:11)

USE	APPLICATION EXAMPLE
Data Collection	Airline Reservations
Remote Job Entry	Local Access to Remote Computing Facilities
Information Retrieval	Credit Checking
Conversational Time-Sharing	Many Simultaneous Users Allowed Interactive Computer Use
Message Switching	Electronic Mail
Resource Sharing	Several Computers Allowed Access to the Same Database, Printer, or Peripheral
Distributed Processing	Several Remote Computing Facilities Cooperating to Solve the Same Problem

computer, this paper considers a network node (UNID II) to be separate from host computers. Node functions include switching, concentration, and buffering and are described in detail immediately following the topology discussions.)

Although five basic network topologies, star (centralized), loop (ring), tree, mesh (distributed), and bus are defined, many networks use combinations and variations of these basic topologies. They may be limited to small local areas or span global distances providing worldwide communication. Examples of long distance networks include the SITA worldwide airlines reservation network and the widely studied Advanced Research Projects Agency (ARPA) network. Small distance local networks such as Xerox Corporation's Ethernet and Net/One by Ungermann-Bass Inc. are becoming increasingly popular. Current users of local networks include General Motors, Citibank, and various government agencies (Ref 15:115). Network computer size may range from large main-frame computers to small microcomputers. User peripherals which have no direct association to a particular computer may be connected to the network.

Star Topology

Since overall control in a star network (Figure 1-1) takes place at the central processor, controlling the network and insuring that the network's resources are fully shared and utilized are easier than in other topologies. The main disadvantage is that each remote site requires its own

communication link. All node-to-node communication must pass through the central processor, and failure of the central processor will disable the entire network.

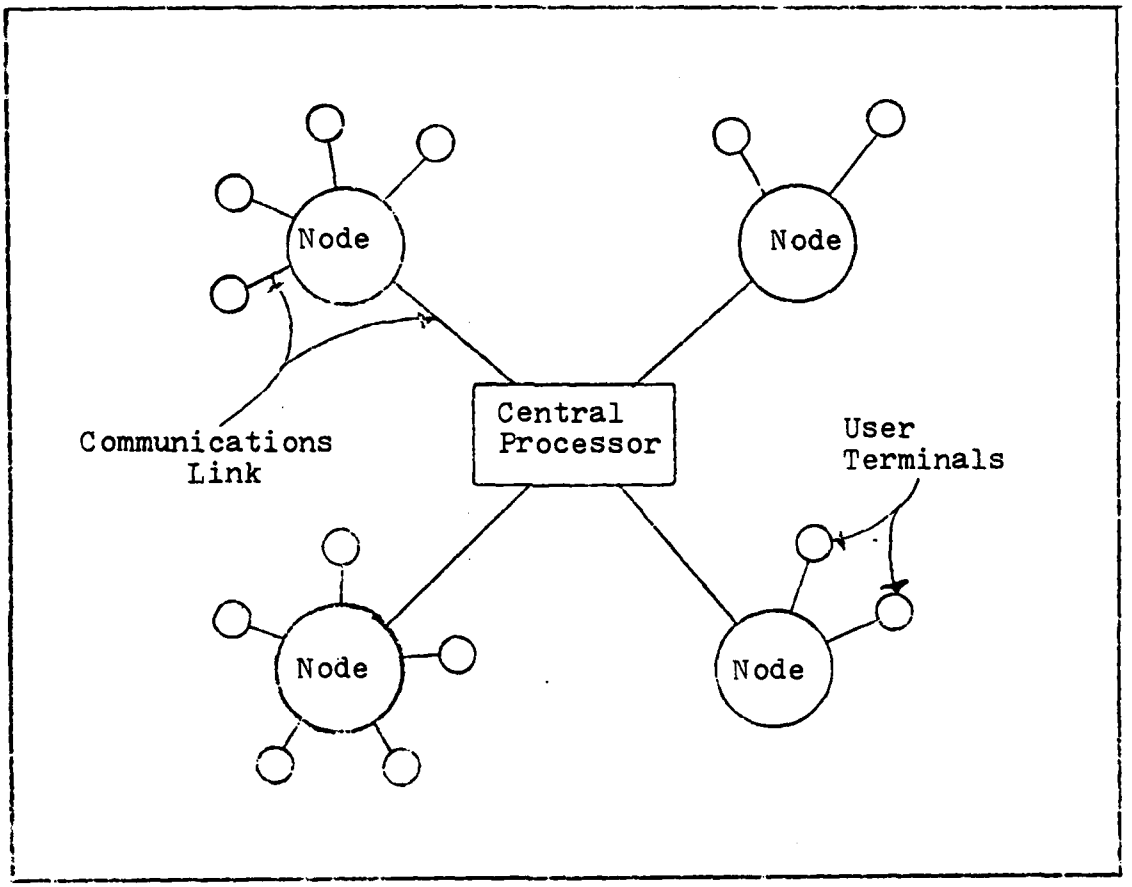


Figure 1-1 Star Topology

Loop Topology

The loop configuration (Figure 1-2) works very well when the nodes are relatively close to each other (Ref 2:113). They minimize the amount of required communication links and are very popular in local networks. Each message circulates around the loop and is repeated by each node

until the message reaches its destination. Since the loop simultaneously carries traffic from many nodes, loop topologies require high capacity transmission links and nodes.

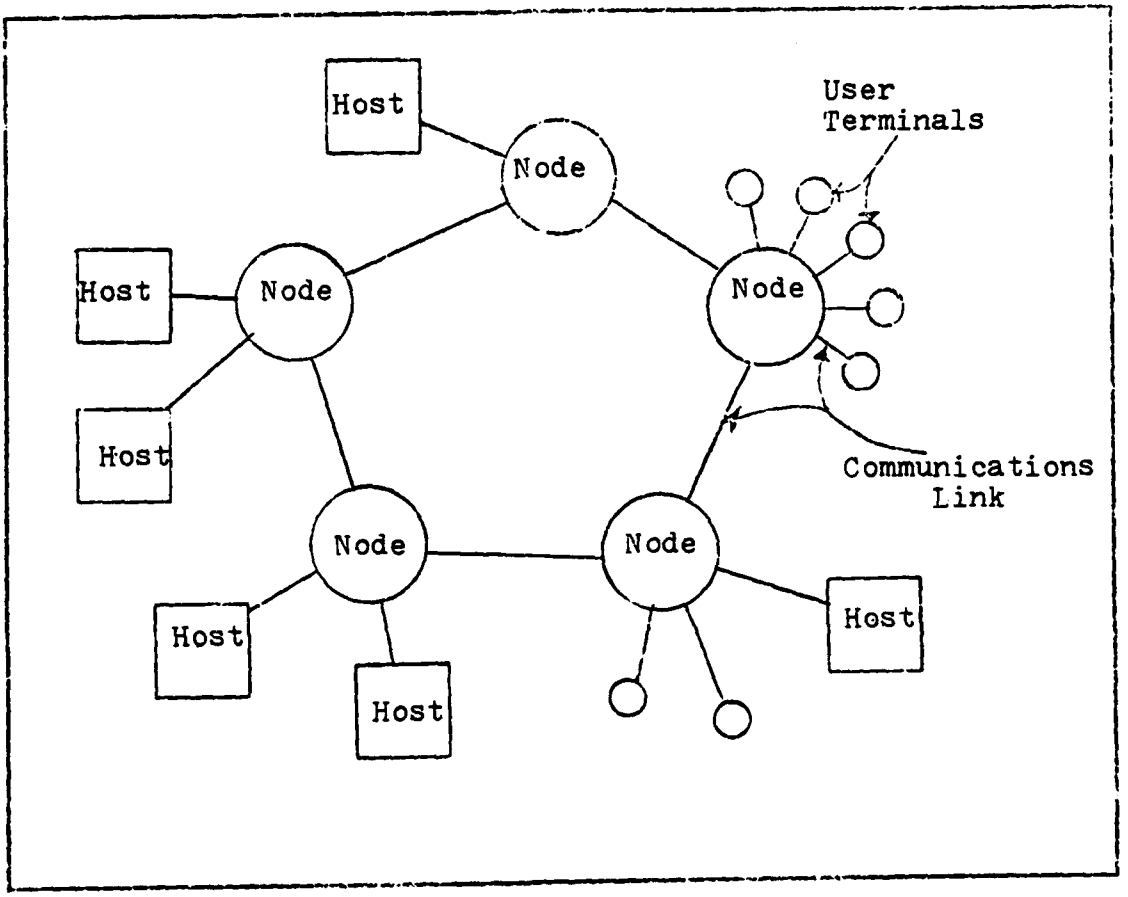


Figure 1-2 Loop Topology (Refs 33:4; 17:84)

The loop concept can be extended to extensive multiloop configurations (Figure 1-3). Each of the loops can act as a relatively self-sufficient building block (Ref 30:564), and with sufficient original planning, the network size can be increased as needed by incrementally adding loops. The costs

of switching nodes and transmission links increase approximately in proportion to the number of added user terminals.

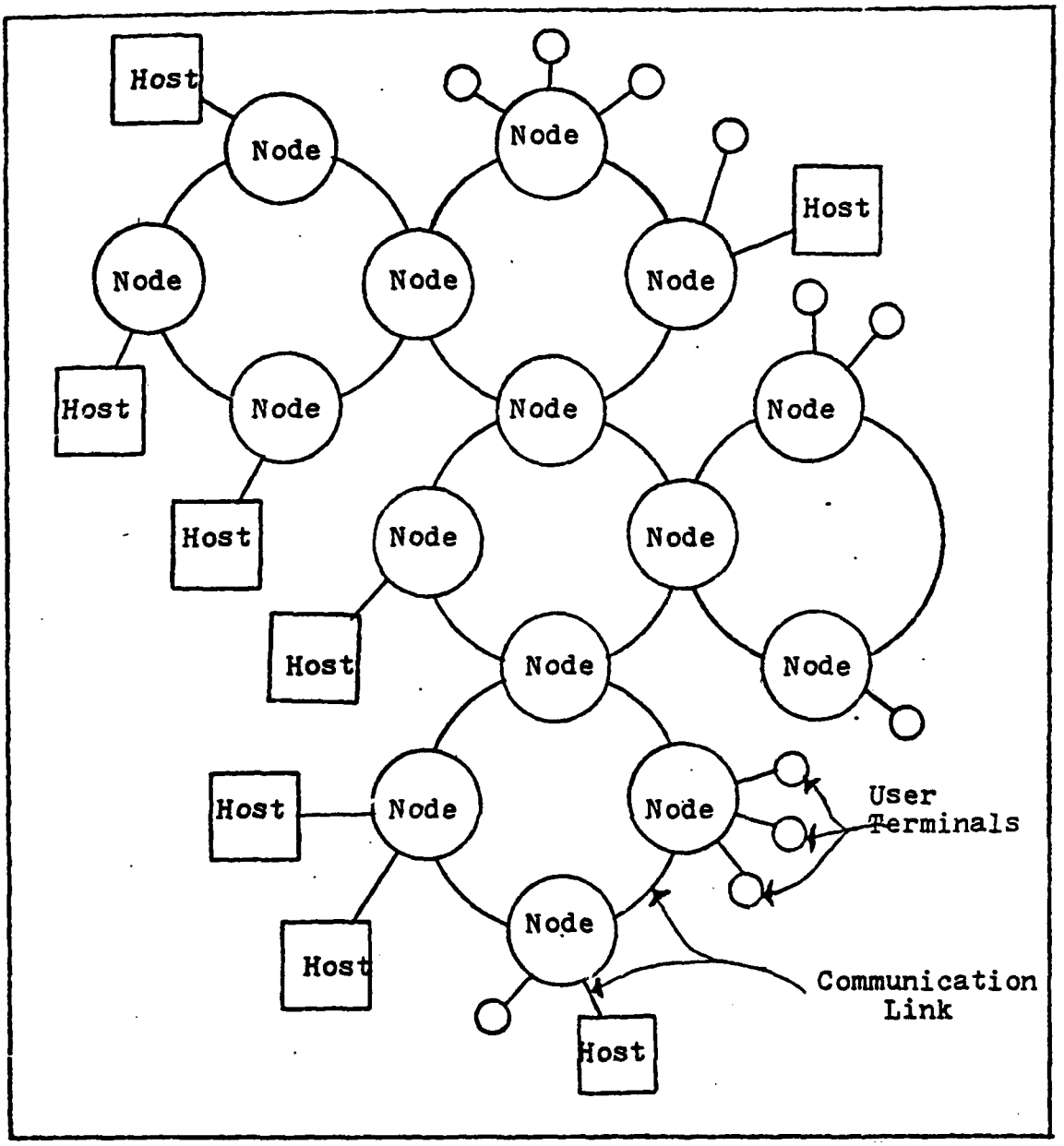


Figure 1-3 Multiloop Configuration

Multiloop topologies have two main advantages; they require no sophisticated common control and the network cost is distributed among the various equipment connected to the network. Since all users on the same loop are connected to the same communications link, communication costs are relatively low. The multiring topology has been recommended for interconnecting computing facilities on a typical Air Force Base (Ref 34:4).

Tree Topology

Similar to a star network, a tree network (Figure 1-4) can be centrally controlled by a main computer. Since communication links close to the central computer are shared, a tree topology requires a lesser net length of communication links than a comparable star network.

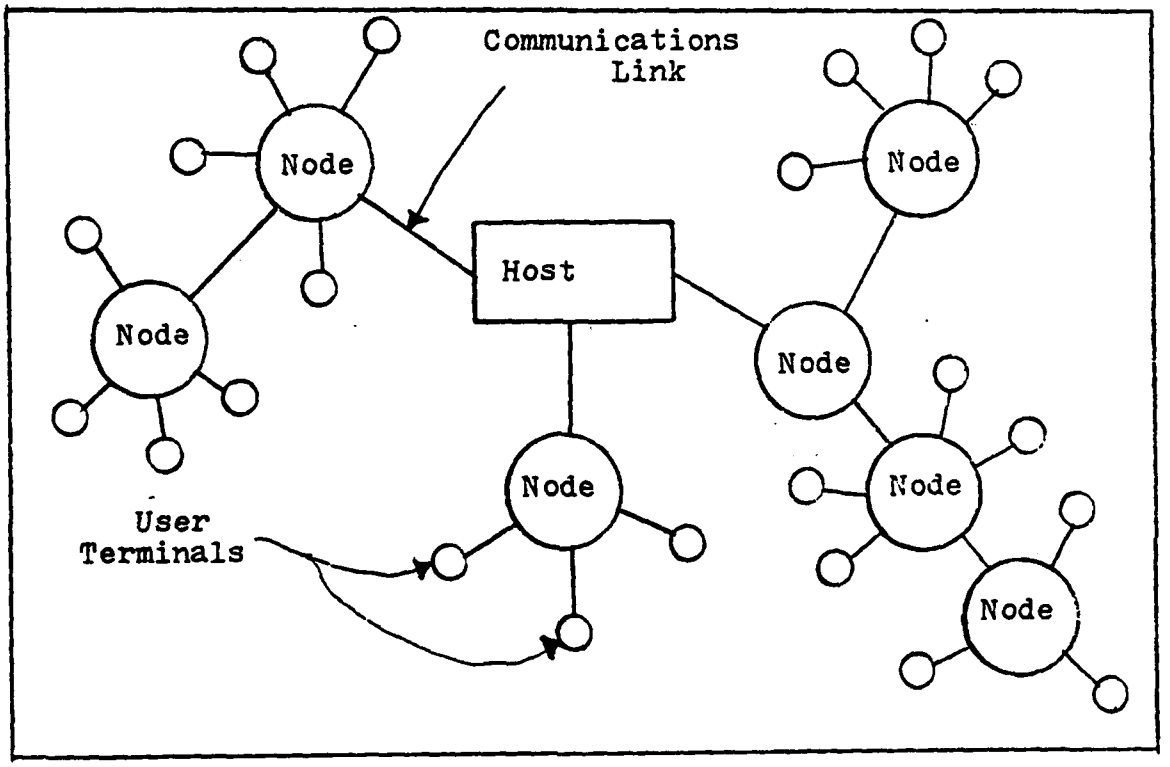


Figure 1-4 Tree Topology (Ref 33:4)

Alternatively, a tree topology may be arranged in a hierarchical structure (Figure 1-5) or in a combination of the centralized and hierarchical structures. In the hierarchical configuration, components high in the hierarchy are often more general while the lower-level components are more specialized (Ref 4:33). Since most processing is done close to the endpoints, data communication costs are kept low and the system is fairly modular. Due to their flexibility, hierarchical tree networks are very popular (Ref 4:32).

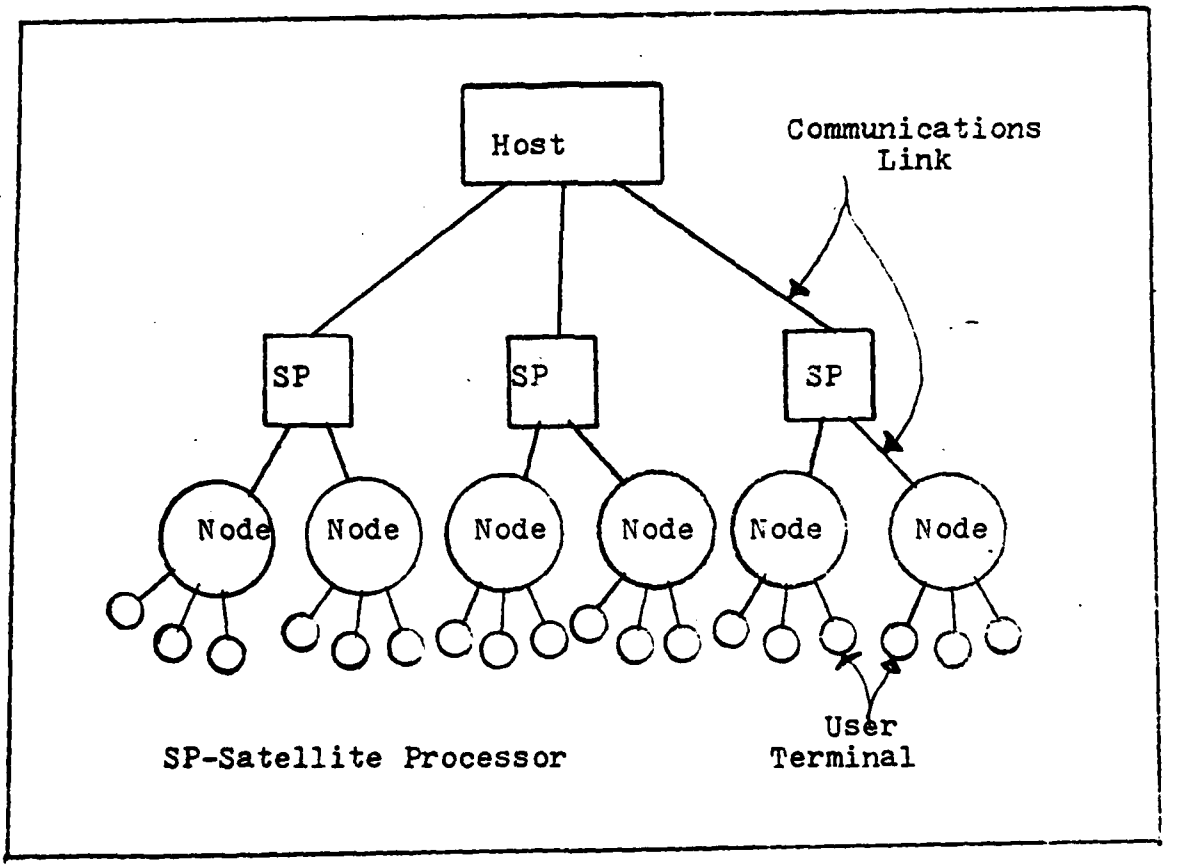


Figure 1-5 Hierarchical Topology (Ref 4:32)

Distributed Topology

Distributed topology (Figiure 1-6) is typical of inter-state networks. Due to their high connectivity, multiple source-to-destination paths often exist. Communication link redundancy increases reliability but makes controlling the network more diffulcult. Nodal interfaces need to be more complex.

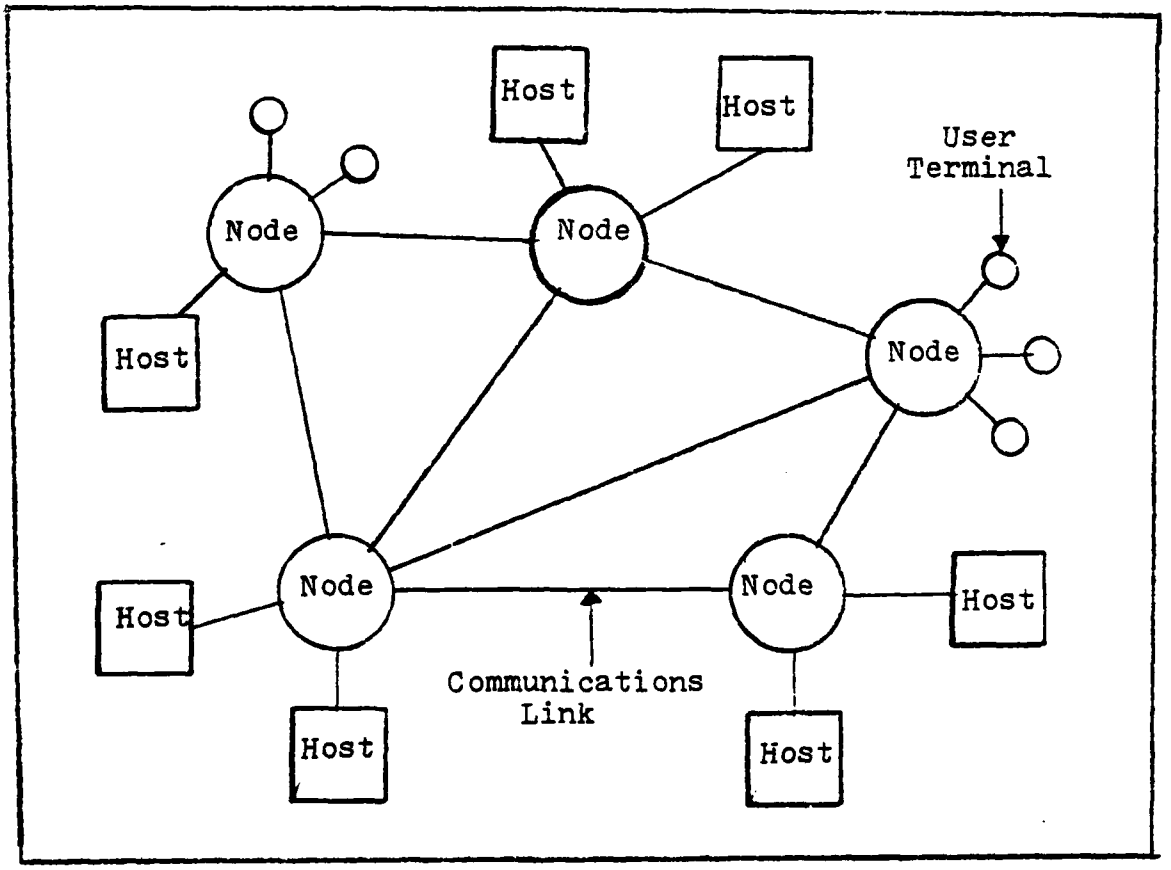


Figure 1-6 Distributed Topology (Ref 33:4)

Bus Topology

Due to their relative simplicity, bus architectures (Figure 1-7) are very popular in local networks. Previously discussed topologies briefly store and retransmit messages at each nodal point along the route from source to destination. In bus topologies, messages are sent by broadcasting over a common transmission channel connected to each network node. Since all messages are broadcast to all nodes, this topology does not allow messages to be specifically routed from one point to another as is possible in store-and-forward topologies.

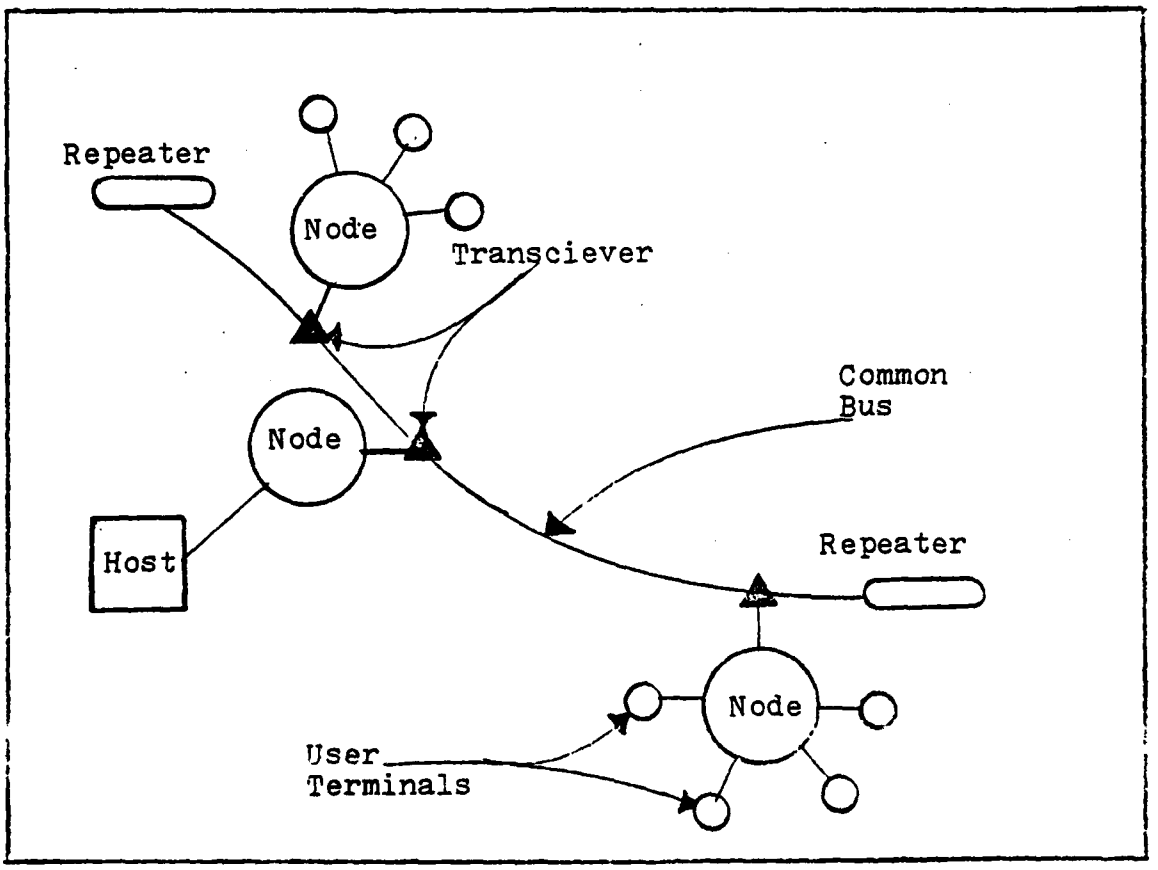


Figure 1-7 Bus Topology (Ref 2:92-96)

Network Node Functions

As a prelude to the Requirements Analysis (Chapter II), the node functions which UNID II will be required to handle are briefly described. The nodes shown in the previous figures perform four main network interfacing functions, (1) concentration, (2) switching and routing, (3) front-end processing, and (4) terminal interfacing. Since nodes handle the actual network information transfer (Ref 40:139), they need to be flexible and efficient. The speed, capability, and reliability of a network is largely determined by the performance of its nodes.

Concentration

Although time-division multiplexing (TDM) could be used, concentrators usually employ asynchronous time-division multiplexing (ASTDM). While TDM alternately allocates each transmission line a time segment for transmission, ASTDM dynamically allocates time segments only to currently active lines. Since TDM may allocate time segments to transmission lines not presently requiring service, ASTDM is more efficient and increases overall throughput. Unlike TDM, ASTDM uses a buffer to allow the input rate to vary from the output rate which smooths message flow and accomodates different transmission rates within the network. Messages are buffered until outgoing links become available and a copy of the outgoing message is often retained until an acknowledgement is recieved that the outgoing message was

successfully transmitted. Concentrators edit messages, format messages according to network protocol, append message headers, and perform code conversion.

Switching and Routing

At each intermediate node, messages are routed to the next node in route to their destination. Routing is often considered part of the concentration function but the GE Information Services Network is an example of a network which uses computers for the sole purpose of directing messages to a particular central concentrator. The GE Information Services Network is the world's largest commercial data processing network and provides service to over 500 cities worldwide (Ref 33:15-16).

Front-End Processors

Front-End Processors are used to interface host computers to the communications network. Even though a general purpose host computer can handle both data processing and communications functions, this approach is often inefficient or economically unattractive. The use of a separate front-end processor prevents communication tasks from seriously degrading the performance of the host. Similar to a concentrator, typical communication tasks include message formatting, editing, and buffering.

Data communication is often bursty meaning that message traffic is very intense for short intervals, and the average time between messages is much larger than the time needed to

transmit a single message. Often, general purpose computers do not handle message traffic efficiently. Network access software does not require the full processing power of a scientific computer yet can consume a substantial amount of the host computer's time and memory space. Using a small separate front-end processor or network-to-host interface as a parallel computer improves the network cost/performance ratio. The network becomes more modular since either the host or front-end processor can be modified or replaced without directly affecting the other device (Ref 1:3-7).

Terminal Interface

The terminal interface allows access to the network, and from one terminal location, the user can access data and use programs that run on various network computers. Similar to other network interfaces, the terminal interface provides data editing, formatting, and error control. Since the user often inputs data slower than the network transmission rate, the terminal interface employs a buffer to temporarily store data until it can be output at the network transmission rate.

Purpose

The purpose of this research effort is to design a general-purpose network interface capable of handling all the functions which were defined in the previous section. These functions include concentration, switching and routing, front-end processing, and interfacing user terminals to the network. UNID II should be capable of

accommodating a wide variety of network topologies and protocols. In the past, most local network interfaces were custom designed for each application (Ref 7:102). To lessen the large hardware and software development costs incurred by custom design, UNID II must be designed with enough flexibility to handle a wide variety of network interfacing needs. UNID II should also use modern architecture to keep energy costs low and allow relatively high data-transmission rates.

A secondary objective is to study a state-of-the-art microprocessor family and incorporate newer technology in the hardware design. Since more than one processor may be required to accommodate UNID II's processing requirements, UNID II will be prototyped using the 8086 microprocessor family whose hardware is designed to support multiprocessing.

Approach and Overview of the Thesis

Chapter I deals with the concept of a network interface and includes discussions describing various node functions and topologies. Since UNID II is designed to handle the different node functions and topologies as defined by these discussions, Chapter I provides background information for Chapter II which defines the functional requirements of a network interface. Chapter II uses the information obtained in an extensive literature search to develop a Data Flow Diagram (Ref 38) model of UNID II. From this model, Chapter III develops the overall hardware design of UNID II.

Chapter IV enhances the information presented in Chapter III. While Chapter III develops UNID II's design as a logical sequence of steps, Chapter IV discusses why the particular hardware configuration was selected rather than a number of other possible configurations. The relative advantages and disadvantages of previous design iterations are discussed. Chapter V presents conclusions and recommendations.

II. Requirements Analysis

The purpose of this research is to design a flexible network interface (UNID II) capable of handling all the general topologies and interfacing functions reviewed in Chapter I. Chapter II uses the background information presented in Chapter I as a basis for determining UNID II's functional requirements.

Requirements Background

A Structured Analysis and Design Technique (SADT) model for the original UNID was developed by Sluzevich (Ref 34). These diagrams provide excellent insight to the functions of a network interface and are recommended for reference. After Sluzevich completed his research in 1978, two more graduate research efforts resulted in a prototype UNID (Refs 3,5). In 1981, William Hobart completed his research effort which designed a local computer network utilizing the UNID developed by earlier efforts (Ref 17). Having the benifit of research following Sluzevich's original report, it seemed worthwhile to re-evaluate the original requirements model.

Requirements Approach

This research develops a set of Data Flow Diagrams (Ref 38) which outline UNID II's functional requirements and provide a basis for the design. The inputs for developing the Data Flow Diagrams are presented in the following sections and consist of input requirements and practical constraints. The input requirements are an expanded defini-

tion of what capabilities a user would expect a network interface to have and were obtained from an extensive literature search.

Requirements Observations

UNID II can be considered a protocol processor which allows network components using otherwise incompatible protocols and transmission rates to operate on the same network. Although UNID II is a programmable device whose functions vary for different applications, UNID II's Data Flow Diagrams provide a general model invaluable for describing essential functions. Specific applications will require additional software functions, but the hardware remains essentially unchanged. Ideally, the software should be layered so only lower-level modules need to be altered for specific applications.

Input Requirements

The primary emphasis of UNID II is flexibility. Since designing custom hardware for each application costs too much money and development time, UNID II is a programmable interface. Depending on its programming, it can be configured to handle a wide variety of interfacing applications. The input requirements outlined in Table II were obtained from an extensive literature search of network interfacing applications and requirements (Refs 17:28; 22:152; 27:3-39; 33:42; 34:12; 35:195; 40:139).

Table II

UNID II Input Requirements

UNID II Input Requirements

I. Interface a wide variety of network components and handle various topologies

A. Accommodate dissimilar computing equipment

1) Accomplish code conversion

2) Perform data-rate speed conversion

B. Interface peripherals and user terminals to network

C. Interface host computers to network

D. Provide a network-to-network interface (gateway)

II. Perform independently of network components

A. Handle network data transmission and reception

1) Accommodate network throughput requirements

a) Provide flow control

2) Adaptable to different protocols

a) Handle both synchronous and asynchronous communication

b) Edit and pack characters into a formatted message

c) Unpack a message

d) Perform Serial to parallel data conversion

e) Handle error control functions such as Message Acknowledge, No Acknowledge, Repeat, and Timeout

3) Have error checking and recovery capability

B. Relieve host computers from network specific functions

1) Provide a buffer to smooth message traffic

2) Poll communication lines if they are multidropped

3) Handle interrupts

4) Route messages to desired destination

5) Collect performance, traffic, and error statistics

Table II

UNID II Input Requirements

Interfacing Capabilities

UNID II should be able to interface a wide variety of different network components. Each component is connected to the network via communication links and an interface is required at every junction of two or more links. Within a single network, the number of interfaces required can be numerous and each interface may be programmed differently depending on the kinds of components interfaced and the amount of of communication links connected to each interface. UNID II should also be capable of handling a variety of different topologies and acting as a network-to-network interface. When dissimilar computing equipment is used, UNID II may be required to perform code conversion before messages are retransmitted out on the network. Since different network components operate at different speeds, data must be buffered and retransmitted at varying speeds.

Transparency Requirements

UNID II should not be dependent on network components to perform its interfacing functions; and ideally, UNID II should be able to perform its communications functions even with one or more host computers completely shut down. UNID II should be capable of handling various data-communication protocols and able to recover from data-transmission errors. It should provide a buffer to smooth message traffic and operate fast enough not to impede the performance of network components. Each node should route messages to the next node.

8086 and 8089 Microprocessors

Since several processors may be required to accommodate UNID II's processing needs and the 8086 family of processors includes bus support circuitry which makes implementing a multiprocessing system relatively easy, the 8086 family was considered a good choice for the UNID II prototype. This design decision has many merits. The 8086 family includes the 8089 I/O processor which can give UNID II the ability to perform high-speed message transfers (Refs 12; 36). Since the 8086 can address one megabyte of memory, UNID II is very capable of relieving hosts from network-specific software tasks. A six-byte instruction queue allows the 8086 to pre-fetch object code instructions and largely eliminates the instruction fetch time (Ref 32:7-33). The instruction queue also allows a longer time for memory devices to recieve an address and return data which allows the use of relatively

inexpensive memories (Ref 20:1-17).

The 8086 is a well supported processor. Hardware support includes specially designed bus controllers, bus arbiters, and slave processors. The wide variety of 8080/8085 support chips is also compatible with the 8086. Software support includes PLM-86, a structured language similar to PASCAL. (The software development process for an 8086/8089 system is overviewed in Appendix A.) Since the 8089 I/O processor can translate code or perform Mask/Compare tests during a DMA transfer, it is ideal for UNID II's message processing functions. (Appendix B discusses 8089 DMA transfers.)

Practical Constraints

Testability

Since UNID II is a complex device, testing is a time-consuming activity. Development of a test plan should proceed in parallel with other UNID II development efforts, and ease-of-testability should be built into UNID II. For debugging software, UNID II's hardware configuration should allow a Monitor program to access all memory areas. Diagnostic programs which exercise critical timing and program paths are also valuable.

Modularity

UNID II hardware should be designed in a modular fashion that permits an error in one subsystem to be isolated from other subsystems. Each subsystem should be relatively independent, and interfaces between subsystems

should be simple.

Economic

The original UNID cost approximately $3,000 (Ref 17:58), and the cost of a microcomputer development system for software support is approximately $10,000. To maximize the advantage of developing an improved network interface (UNID II), the cost should be kept within these limits.

UNID II Model

This section presents and explains the Data Flow Diagrams (DFDs) which describe UNID II's functional requirements. DFD's are a graphical tool which use labeled circles (transforms) and arrows to model the logical flow of data in a system (Ref 38:55). Arrows represent data streams while transforms convert input data streams into output data streams. Files or data bases are represented by a straight line (Figure 2-1).

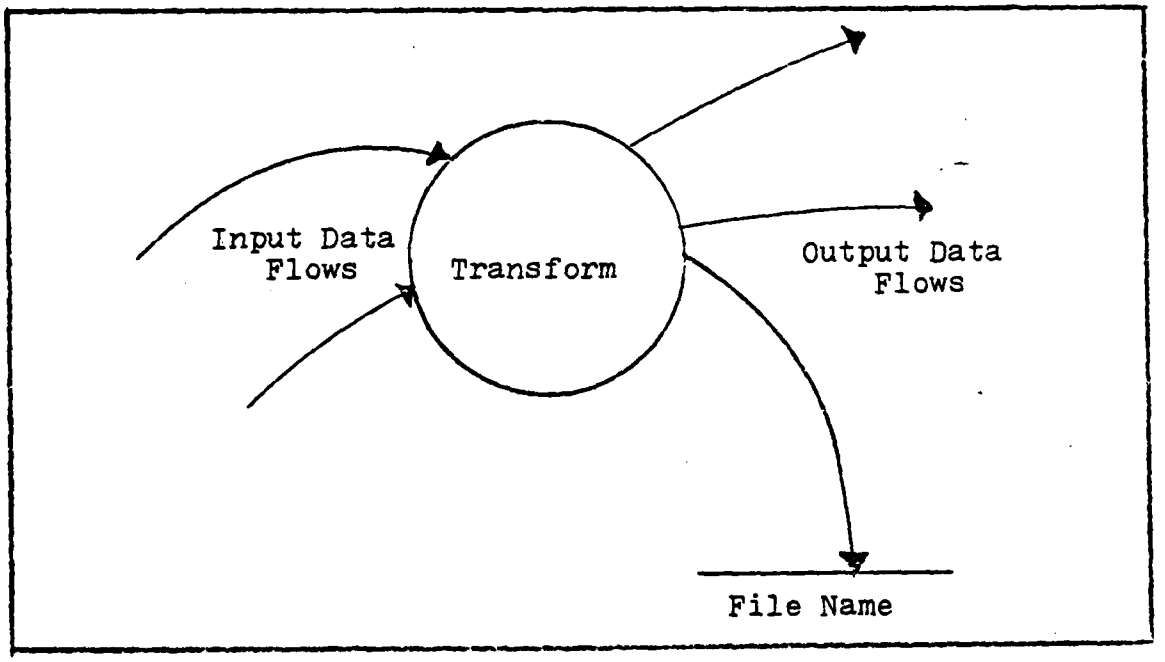


Figure 2-1 Data Flow Diagram Symbols

Data Flow Diagrams are done in levels; the first diagram presented is an overview of the system while successive lower-level diagrams give a more detailed description of a selected transform in the next higher-level diagram. The process of of developing lower-level diagrams (decomposition) continues until each transform or process has been described in sufficient detail.

Table III is an outline for UNID II's DFDs and shows that each transform of the parent DFD was decomposed one level. This amount of detail is sufficient to serve as a guide for selecting UNID II's hardware configuration. To avoid making UNID II application dependent, lower-level functions which represent application-specific software are specified by the user.

Transform Number	Transform Title
0	UNID II Overview
1	Input Local Information
2	Format According to Outgoing Protocol
3	Transmit Network Message
4	Input Network Information
5	Transmit Local Message

Table III UNID II Data Flow Diagram Outline

UNID II connects local network components to the rest of the network. Since local devices rarely use the same protocol as the inner network, Figure 2-2 shows two separate

link control protocols, local and network. A network may be divided into two conceptual areas, an inner area which must follow network protocol and an external area where a local protocol is used. If UNID II is used as a gateway or network-to-network inerface, two different network data-transmission protocols are supported rather than a network and local protocol.

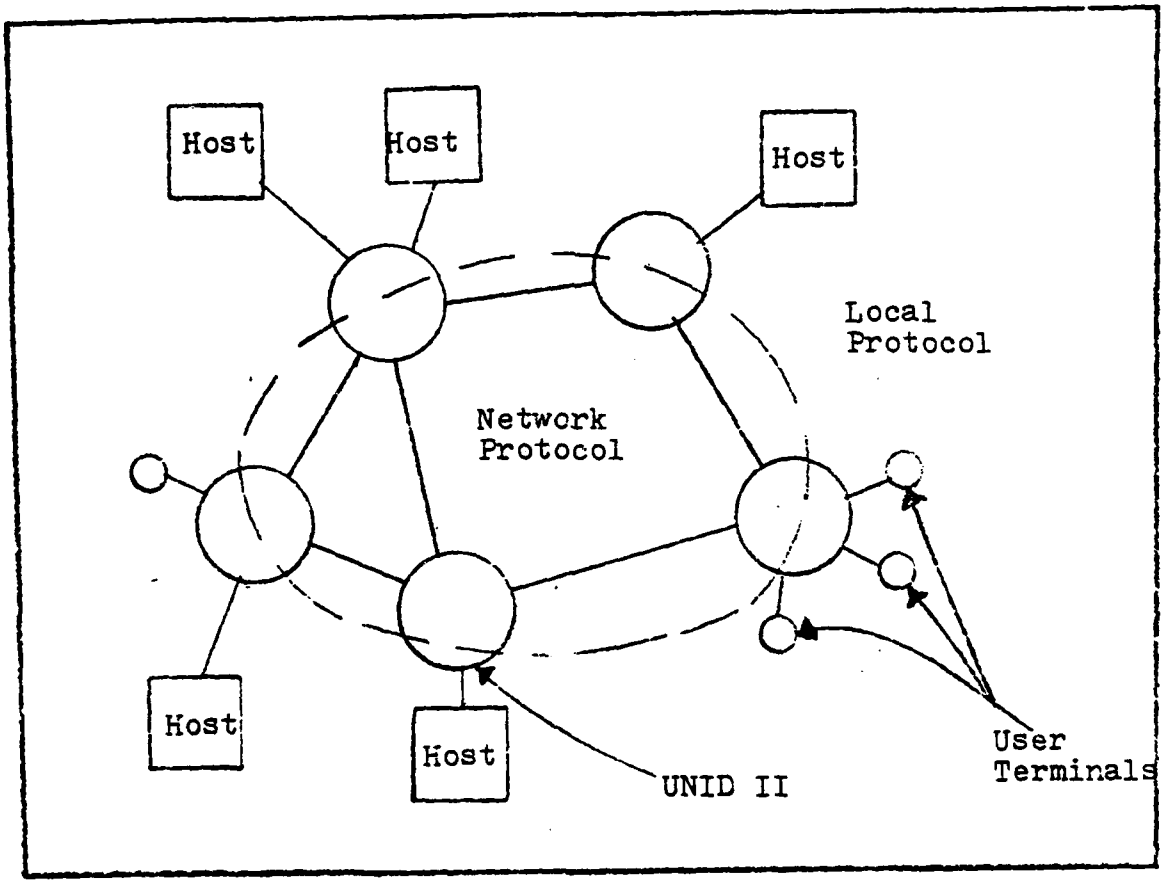


Figure 2-2

UNID II Supporting Both a Network and Local Protocol

UNID II Overview (Figure 2-3)

Messages are processed in three stages; they are input into UNID II from either a local or network source (transforms 1 or 4), edited and formatted according to the outgoing protocol (2), and transmitted to their next immediate destination (3 or 5). The Format-According-to-Outgoing-Protocol function routes messages to their next local or network destination , and allows several local devices connected to the same UNID to communicate without accessing the inner network.

Input Local Information (Figure 2-4)

The Input-Local-Information process places incoming local messages into memory. As information flows into UNID II, control information such as Start/End-of-Information, frame check, and parity check must be distinguished from message text. If a transmission error should be detected, an error message is reported to the local component which sent the message. UNID II monitors incoming links to determine when they begin transmitting and converts incoming messages into bytes or words compatible with UNID II's memory. Incoming messages are buffered, scanned for correction characters, and edited. When the message which is identified by starting address and length is moved into the processing queue, the previously used input buffer space is deallocated. Routing information such as source and ultimate destination is determined and sent to the processing queue as part of the message.

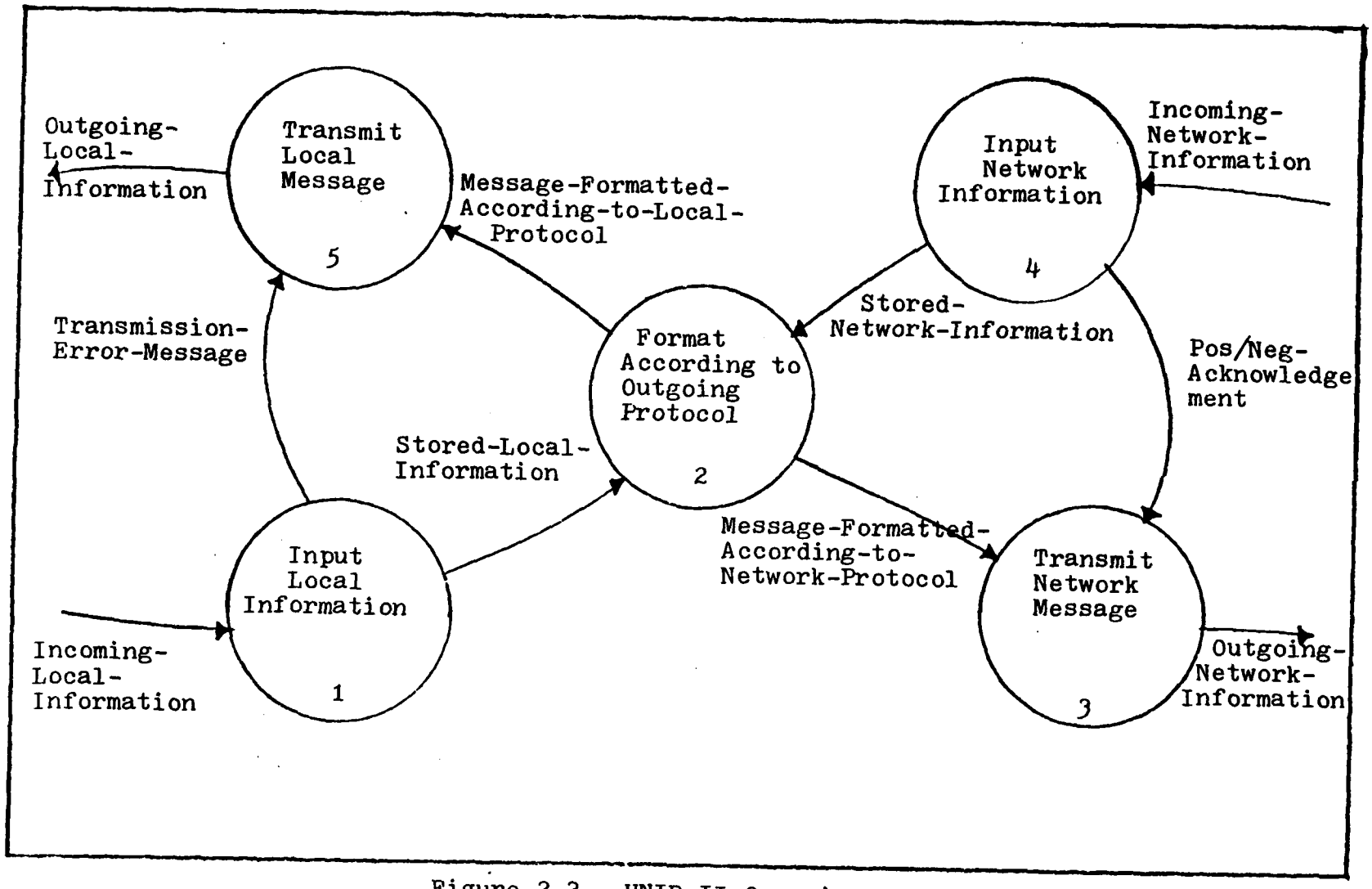


Figure 2-3 UNID II Overview

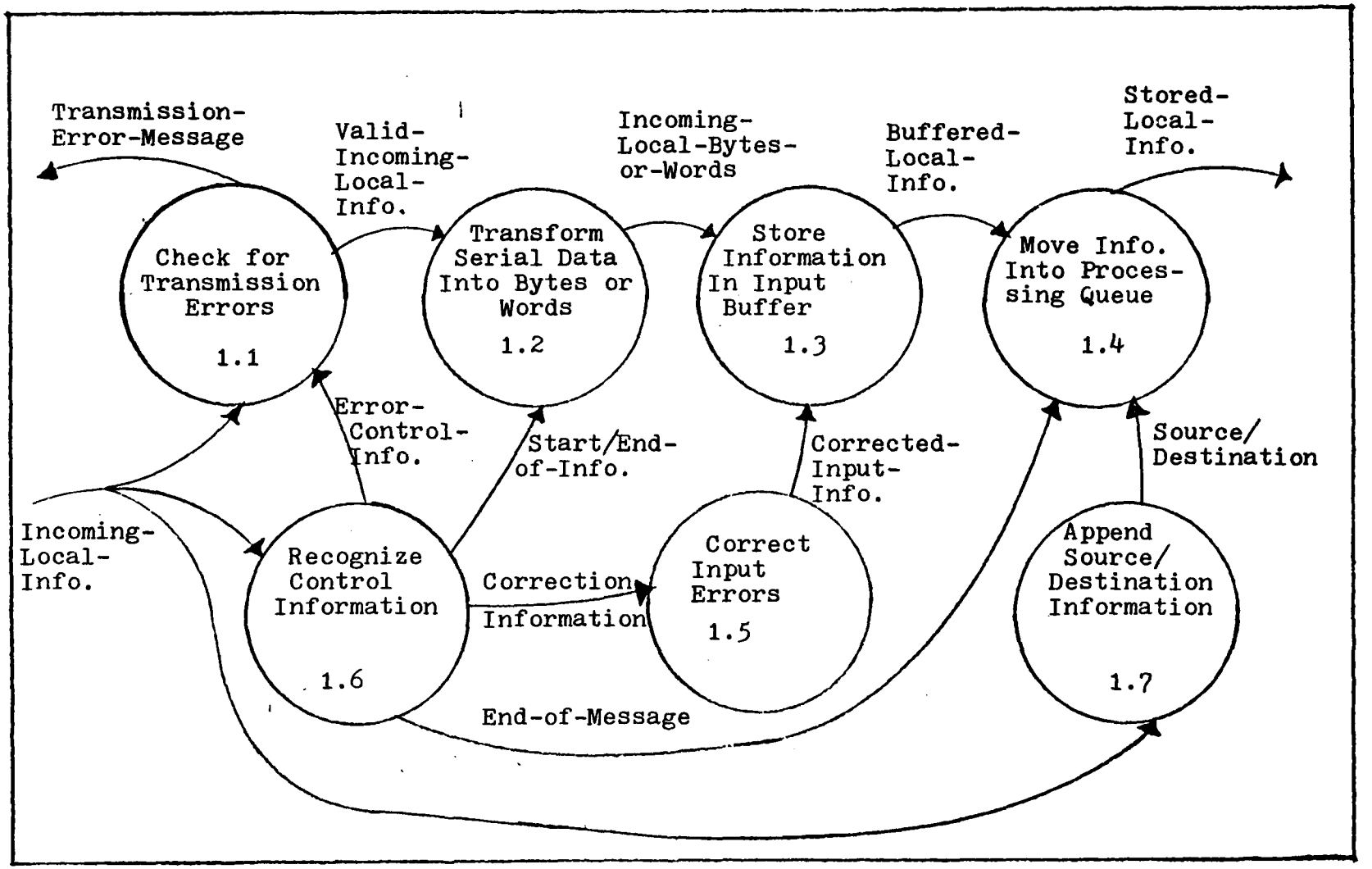


Figure 2-4 Input Local Information

Format According to Outgoing Protocol (Figure 2-5)

Stored network and local information is edited and formatted according to outgoing protocol rules which are stored in UNID II's memory. Since incoming message frames may be interleaved with frames from other messages, the Assemble-Frames function (2.1) is needed to recognize the frames which belong to each message. Incoming local information may use a different code than the rest of the network and require translation. The Determine-Next-Destination function (2.4) uses a file of routing rules to determine next destination information which is appended to the message.

Transmit Network Message (Figure 2-6)

Once messages are formatted according to network protocol, they are moved into the output buffer where error control information is generated and appended to the message. When the appropriate outgoing link is available, the message is transformed into a serial bit stream and transmitted according to outgoing protocol. For the longer distance network links, parallel links were considered to be too expensive since they require a separate transmission channel and associated hardware such as line drivers and recievers for each parallel bit. Transmit-According-to-Outgoing-Protocol (3.7) sends the message at the correct speed and inserts synchronization bits or other information required by the outgoing protocol. Message acknowledge or no-acknowledge signals are used to determine if a message needs to be retransmitted.

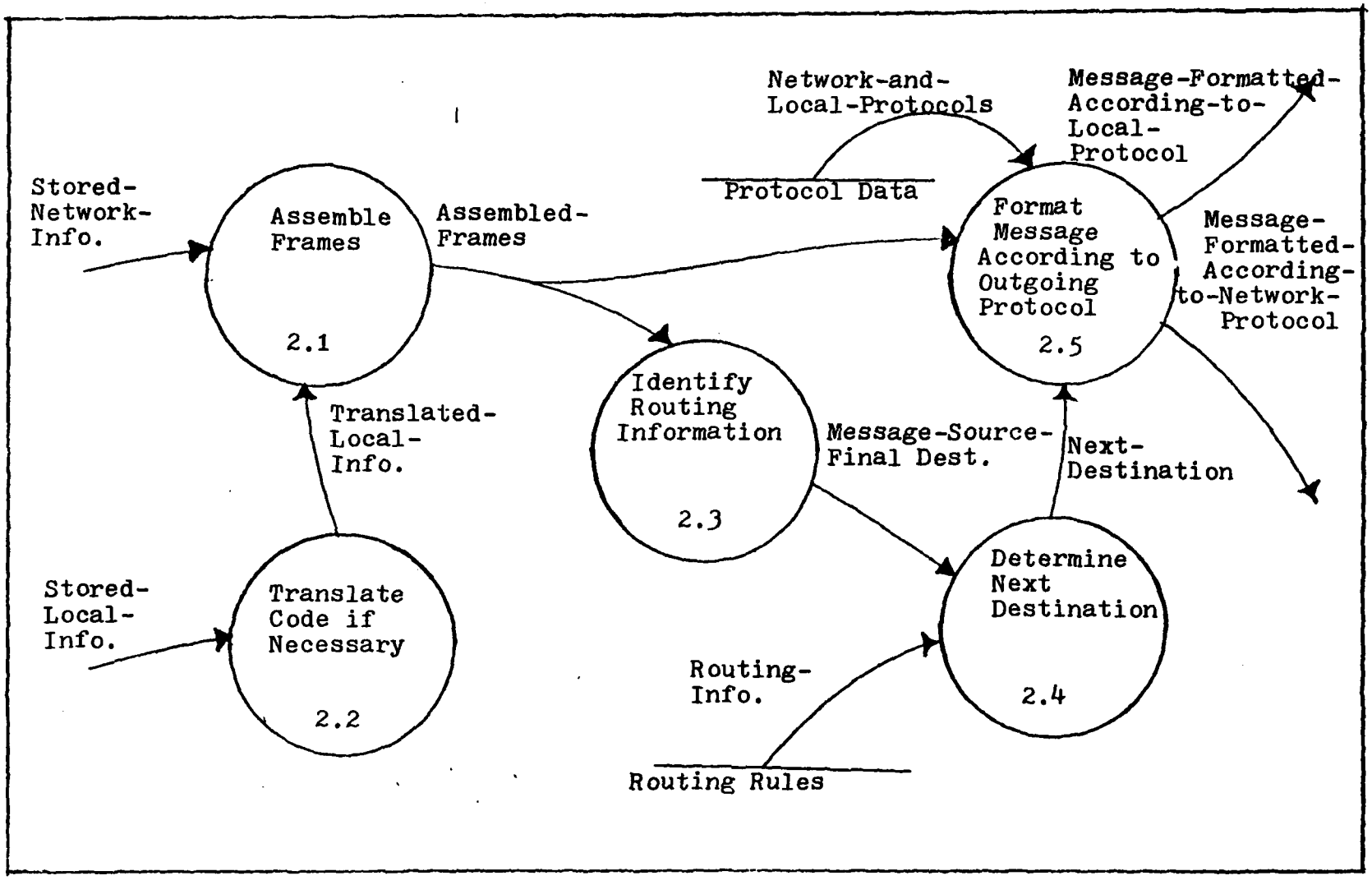


Figure 2-5 Format According to Outgoing Protocol

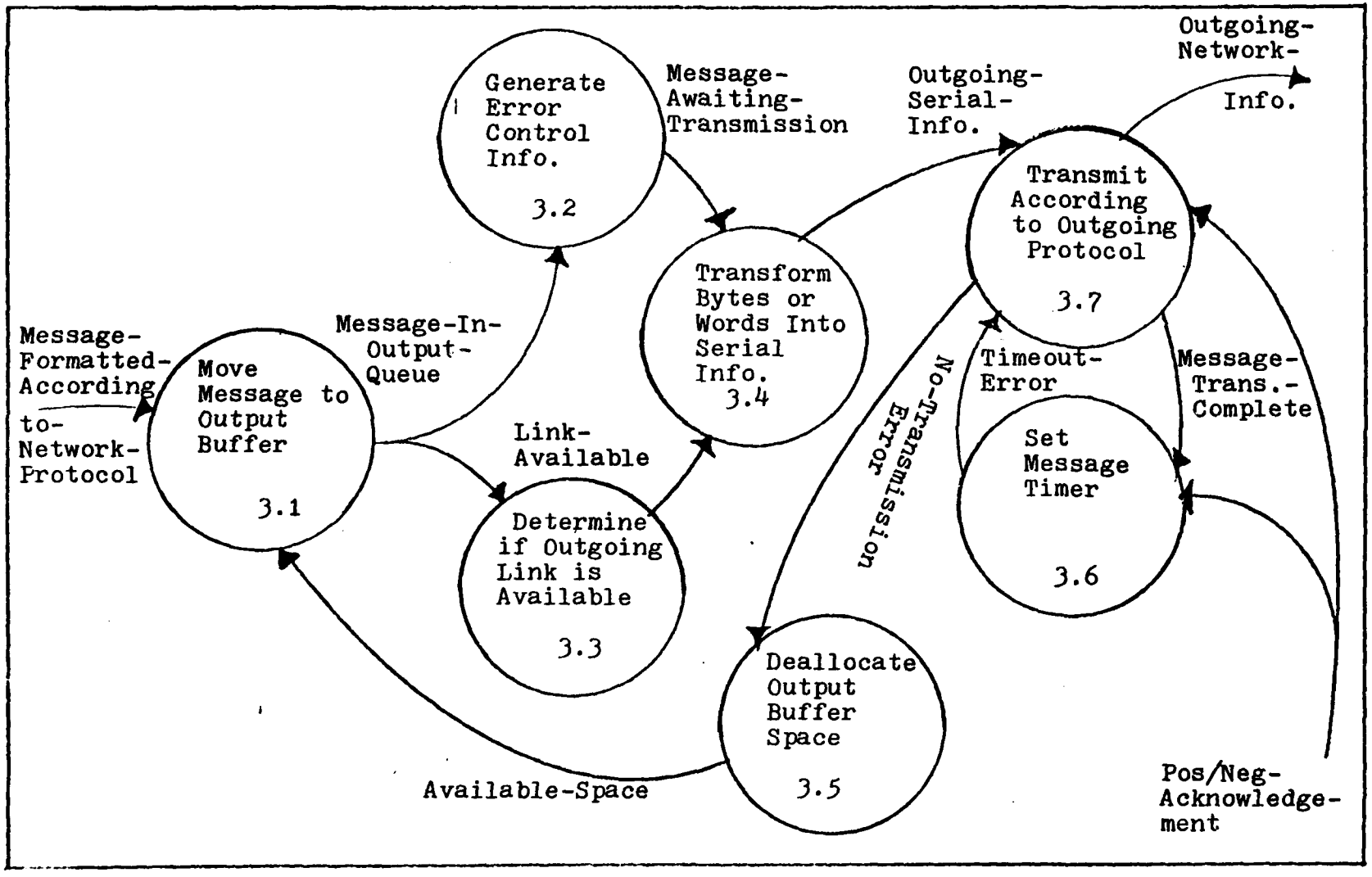


Figure 2-6 Transmit Network Message

Input Network Information (Figure 2-7)

The error control information sent along with the incoming network information is used to detect transmission errors. Depending on network protocol, positive or negative acknowledgement messages are forwarded to the sending node to indicate whether or not the message was successfully recieved. The serial bit stream is converted into bytes (or words) which are placed into an input buffer. Routing information such as message source and ultimate destination is transferred along with the message to the processing queue.

Transmit Local Message (Figure 2-8)

The Message-Formatted-According-to-Local-Protocol is moved to the output buffer allowing the space previously occupied in the processing queue to be deallocated. Error control information is generated and; if the outgoing link is serial, the parallel information contained in UNID II's memory must be transformed into a serial bit stream. Routing information contained within the message is used to determine the recieving peripheral which must be monitored to determine when it is available. Transmit-According-to-Local-Protocol sends the message at the correct speed and inserts start/stop bits or other information required by the outgoing protocol.

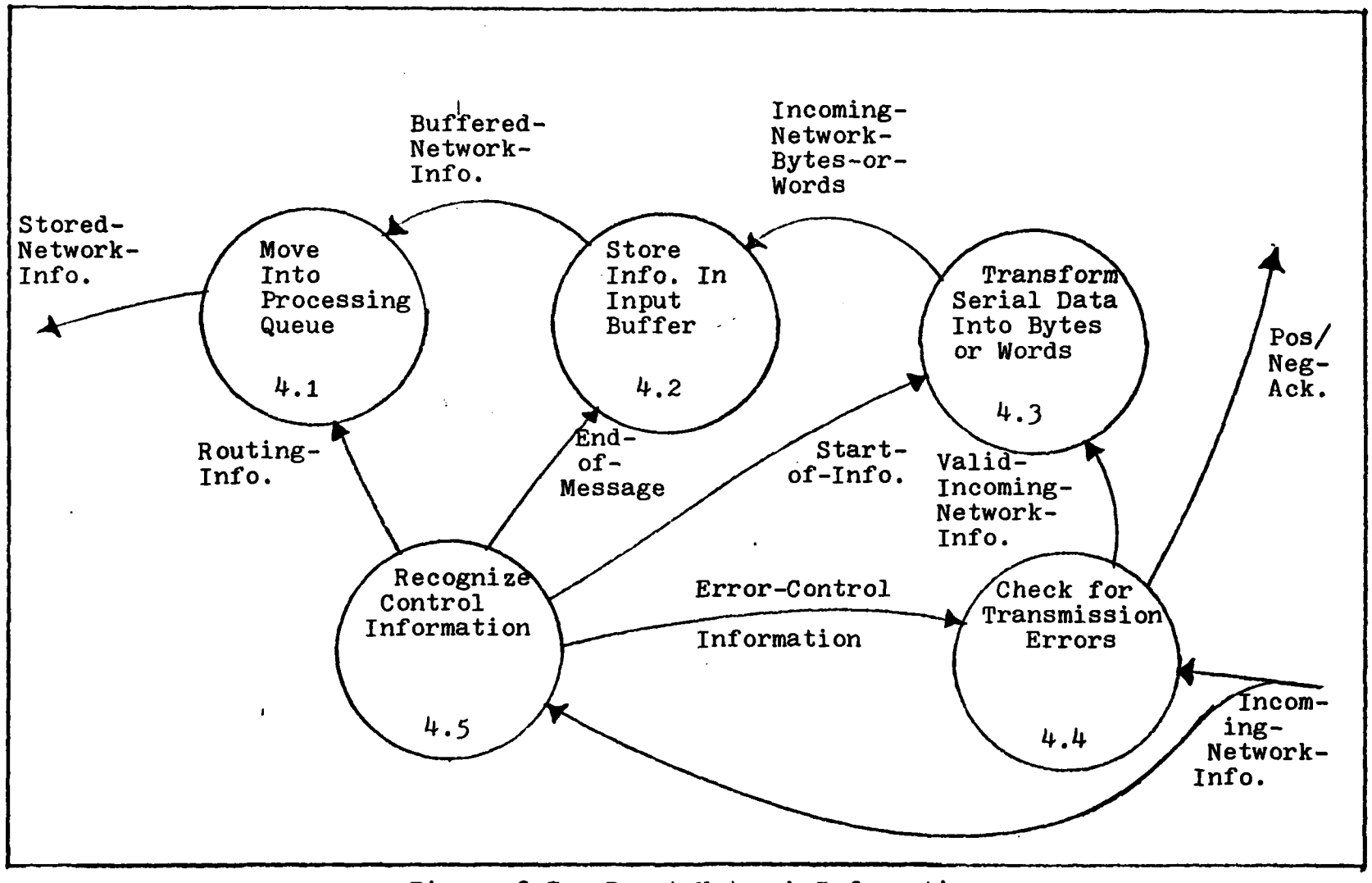


Figure 2-7 Input Network Information

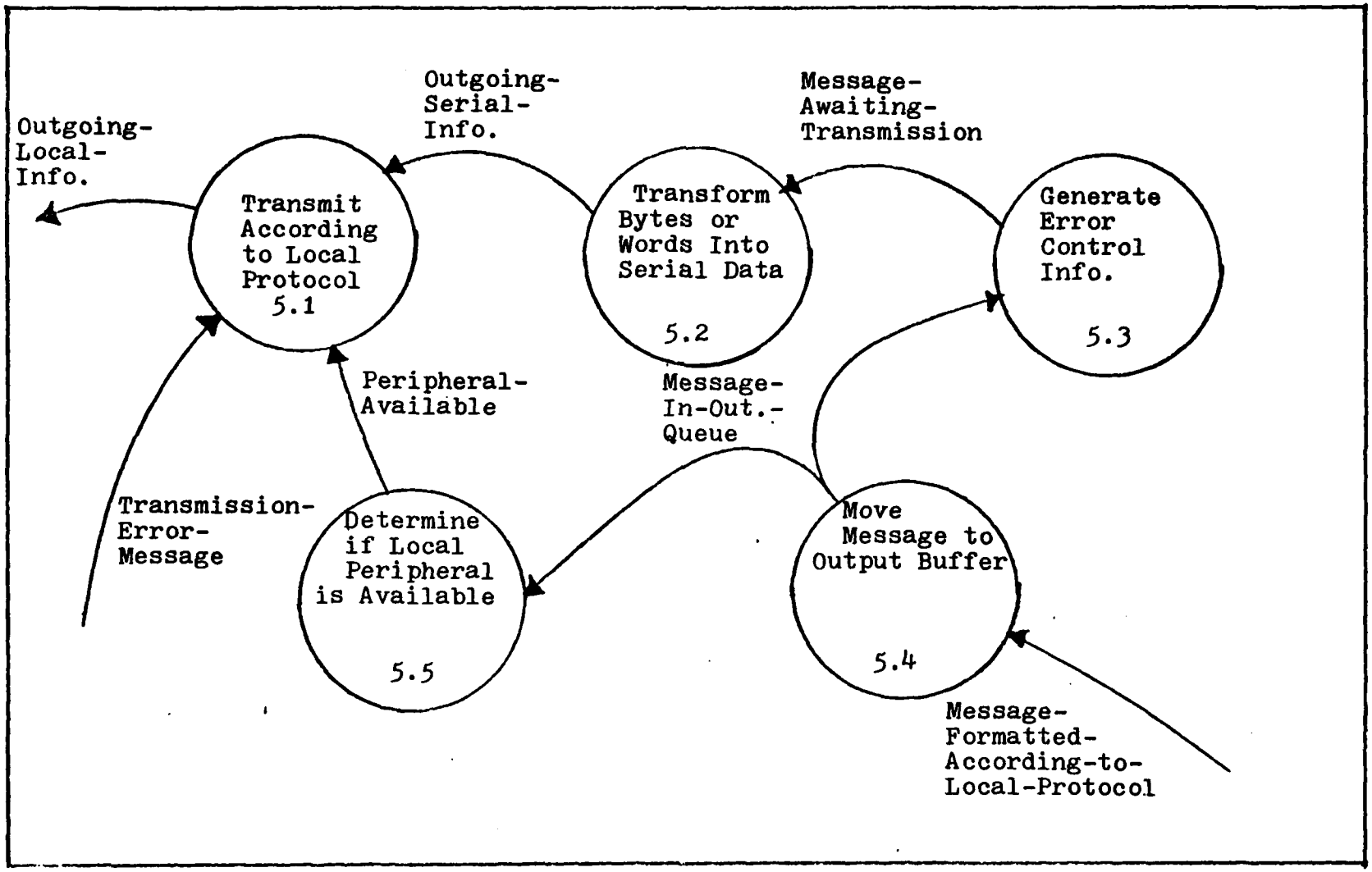


Figure 2-8 Transmit Local Message

Requirements Analysis Summary

The requirements analysis began with a list of input requirements (Table II) which were obtained from a literary study of network interfacing functions and applications. Using Data Flow Diagrams (Ref 38), these input requirements were used to generate a functional requirements model for UNID II. This model categorizes UNID II functions into two main groups; one group of functions handles Local messages while the other group handles Network messages. The functional requirements developed in this chapter will be used as a basis for developing the design of UNID II in following chapters.

III. UNID II's Hardware Design

The most critical design decisions affecting UNID II's cost and performance are the allocation of each functional requirement to either hardware or software components. Since the capabilities of available components should be determined before establishing function partitions, this chapter examines the basic hardware configurations supported by the 8086 microprocessor family and determines which features should be used to maximize UNID II's performance.

Basic Hardware Configuration

The basic hardware configuration supported by the 8086 CPU and 8089 I/O processor (IOP) is shown in Figure 3-1 (Ref 36). Since the I/O subsystem allows data to be input into an I/O buffer without use of the system bus, UNID II's word storage function may proceed in parallel with CPU processing. The 8089 has two address spaces, a one megabyte system space shared with the CPU and a private 64K byte I/O space. Both memory and I/O ports may be placed in either system or I/O space, but the system bus responds to memory commands while the I/O bus responds to I/O commands. A tag bit is used to distinguish between memory and I/O addresses. The IOP fetches data from a source and transfers it to a destination identified by an address which allows data transfers between the system bus and I/O bus, between components on the I/O bus, or between components on the system bus.

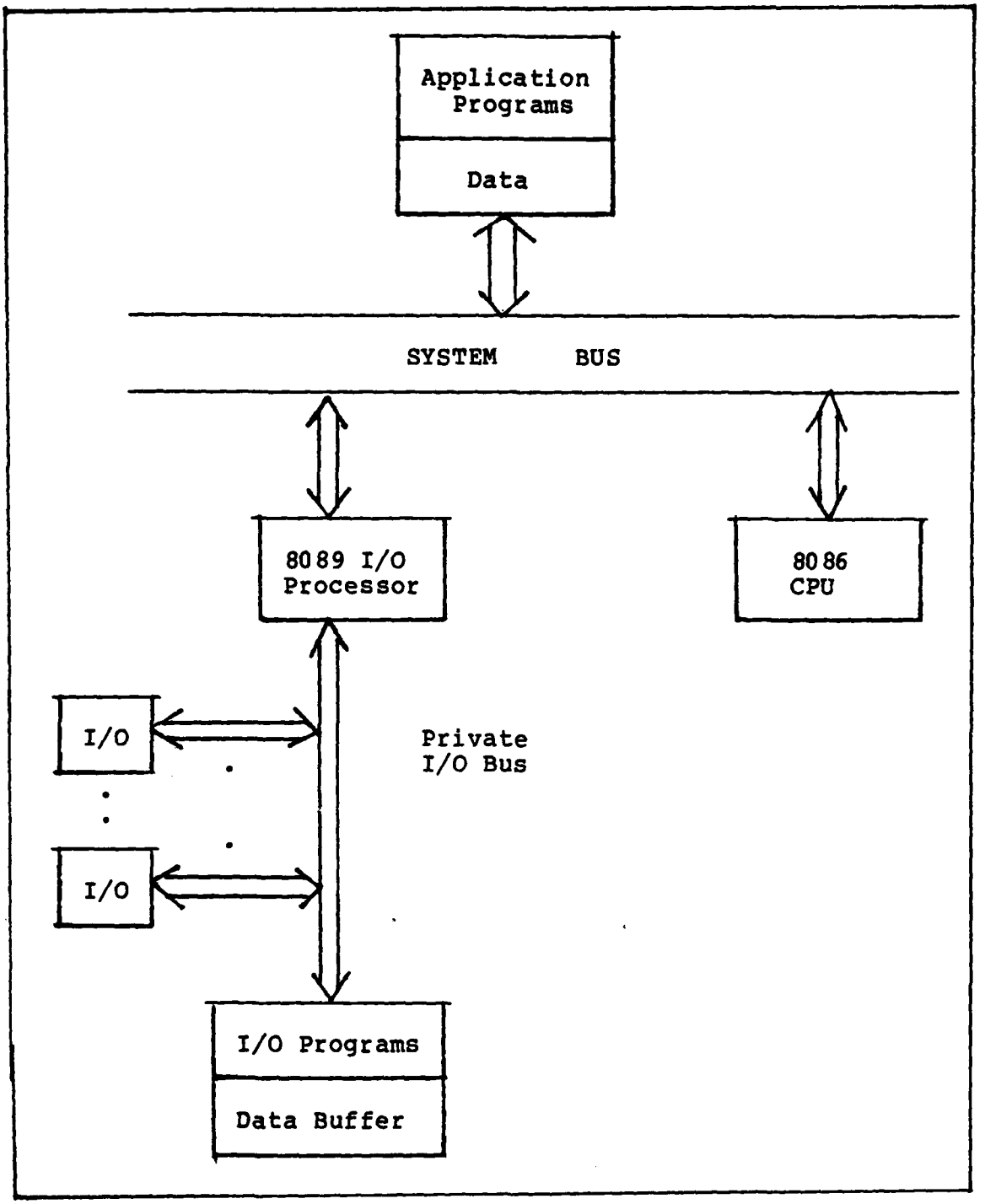


Figure 3-1

Basic 8086/8089 Processor Configuration

Private CPU Bus

The 8086 CPU is normally limited to I/O instructions when referencing I/O space, but some additional hardware can give the CPU its own private bus. Intel literature calls a private CPU bus a resident bus (Ref 18:A-127). Since two bus controllers and two sets of bus transcievers are used to support this configuration, both memory and I/O commands may be used on either the system or resident bus. An EPROM or decoder is used for address mapping which allows only the resident bus to access a portion of the system address space. Similar to the I/O subsystem, the resident bus gives the CPU its own private resources creating a modular, relatively self-sufficient CPU subsystem.

For UNID II, the extra hardware required to implement a resident bus was considered well worthwhile. This configuration frees system memory from applications programs which reduces bus contention and allows system memory to be used soley for interprocessor communication and the transfer and buffering of network messages. Since both the CPU and I/O subsystems have memory space for applications programs unaccessible by the other subsystem, errors in one subsystem are less likely to affect the other subsystem. The ability to isolate errors is considered to be a critical part of helping to satisfy UNID II's requirement for testability. A generalized block diagram demonstrating the concept of a CPU resident bus is shown in Figure 3-2.

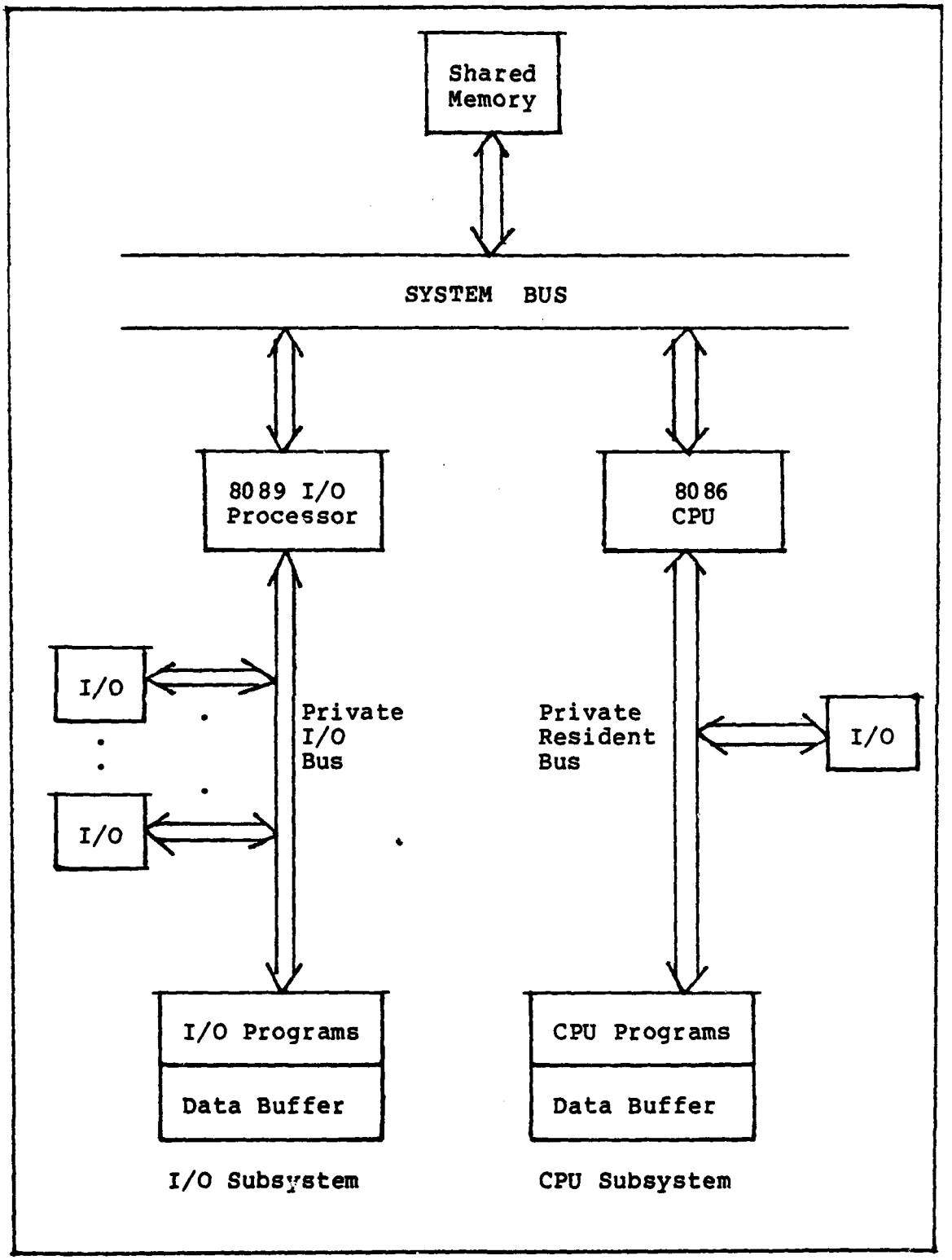


Figure 3-2 8086 CPU With Resident Bus

Another CPU Local to the I/O Processor

Although the 8089 I/O processor is well suited to high-speed data transfers and may operate independently, it lacks many general-purpose data-processing instructions. It may not be capable of handling all UNID II's functional requirements for editing and formatting messages and for generating and verifying error control words. The variety of instructions available in the high-speed network I/O subsystem may be increased by allowing another CPU to share the same local or private bus as the I/O processor (Figure 3-3). This configuration was chosen to allow the network I/O subsystem to completely process messages without accessing the system bus. In the local mode, one processor is idle while the other is active, and the IOP acts as a slave processor which increases the number of instructions available to the local CPU giving the network subsystem specialized I/O instructions. Together, the 8086 and slave 8089 IOP act independently of the system bus. Since 8086/8089 architecture uses logic contained within the processor chips to handle local arbitration, no additional arbitration circuitry is required to allow a CPU to share the 8089 IOP local bus.

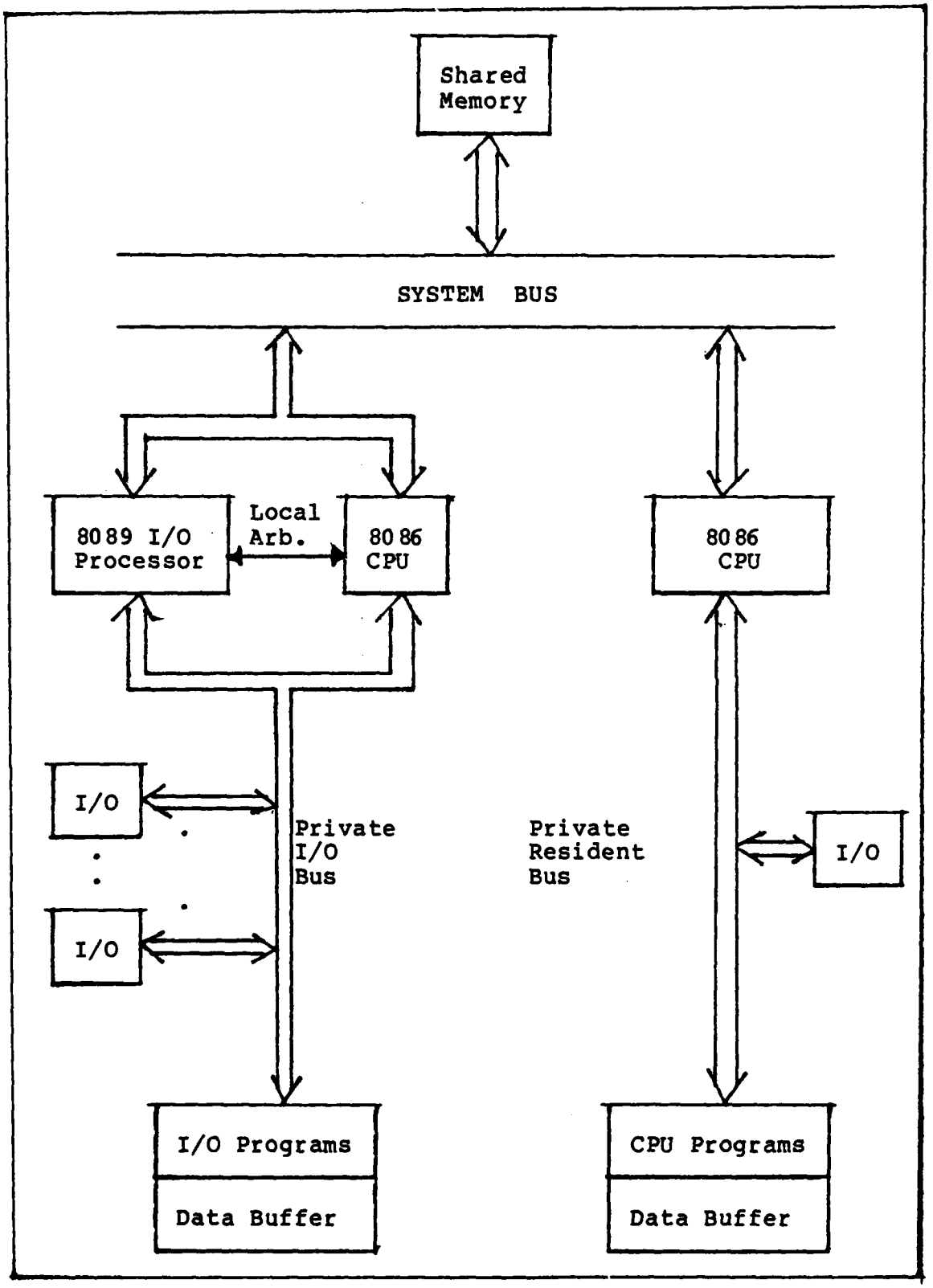


Figure 3-3 An 8086 CPU Local to the I/O Processor

UNID II's Overall Design

Figure 3-4 is a block diagram of UNID II's final prototype design which consists of two independent subsystems that communicate through a block of shared memory. The high-speed Network I/O subsystem handles Network communication links whose service is time-critical while the Local I/O subsystem handles lower-speed local or peripheral links. To allow UNID II to be implemented as quickly and easily as possible, commercially obtained cards were used for the Local Subsystem. Since the control signals the 8086 processor family uses for multiprocessing are identical to Intel Multibus control signals, the Multibus structure was chosen for UNID II's shared system bus.

For flexibility, an interchangeable I/O card is allowed to connect directly to the shared Multibus. Since the I/O card is part of the Local Subsystem, the I/O card should connect to a Local Subsystem private bus. The I/O card connection to the Multibus is an implementation compromise caused by the use of commercial cards; but to reduce bus contention, future modifications may include placing all Local I/O on a Local Subsystem private bus.

Network I/O Subsystem

Ideally, the Network I/O subsystem should also be implemented using a single board computer, but no commercial cards containing 8089 I/O processors are presently

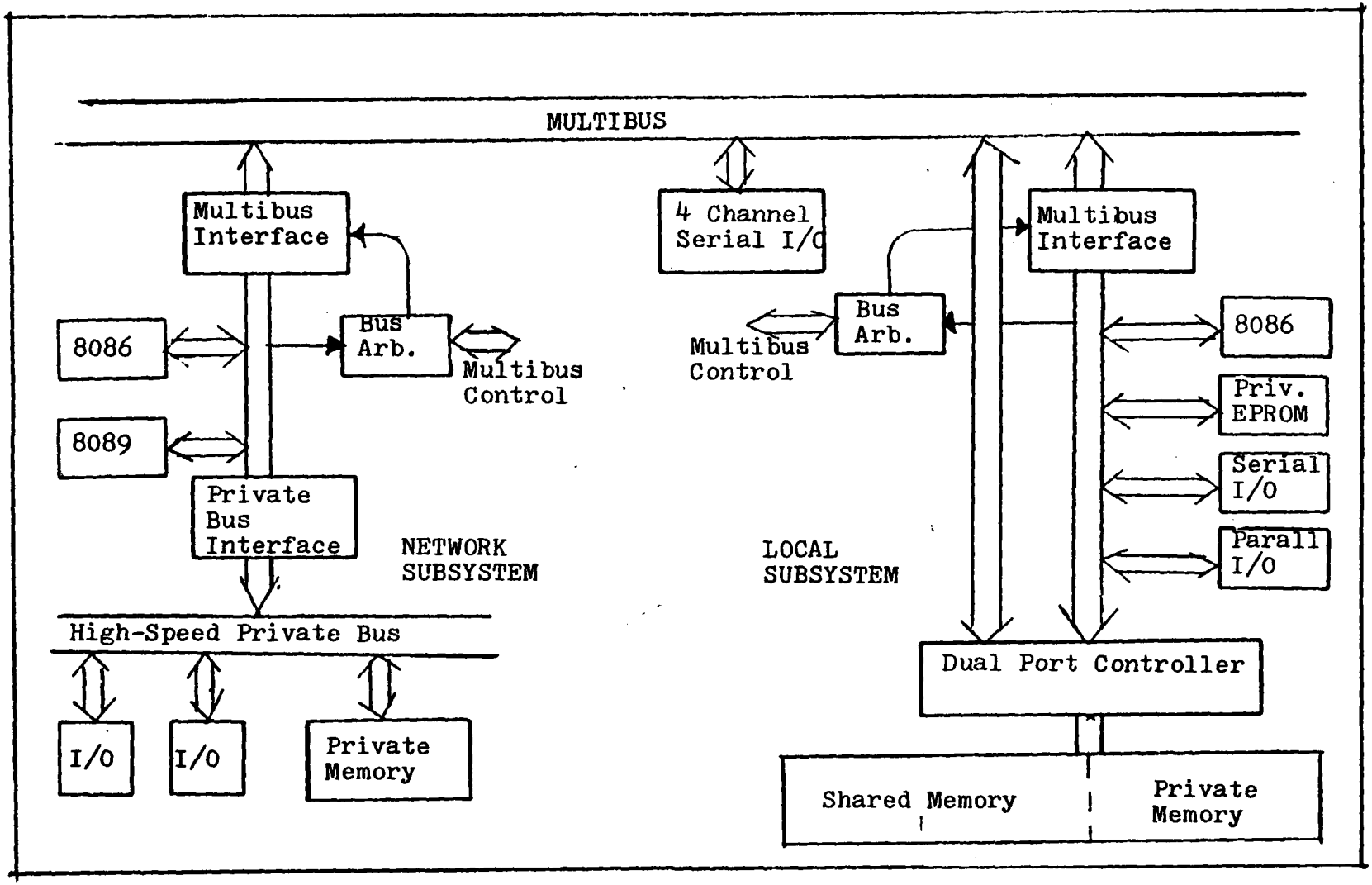


Figure 3-4 UNID II Block Diagram

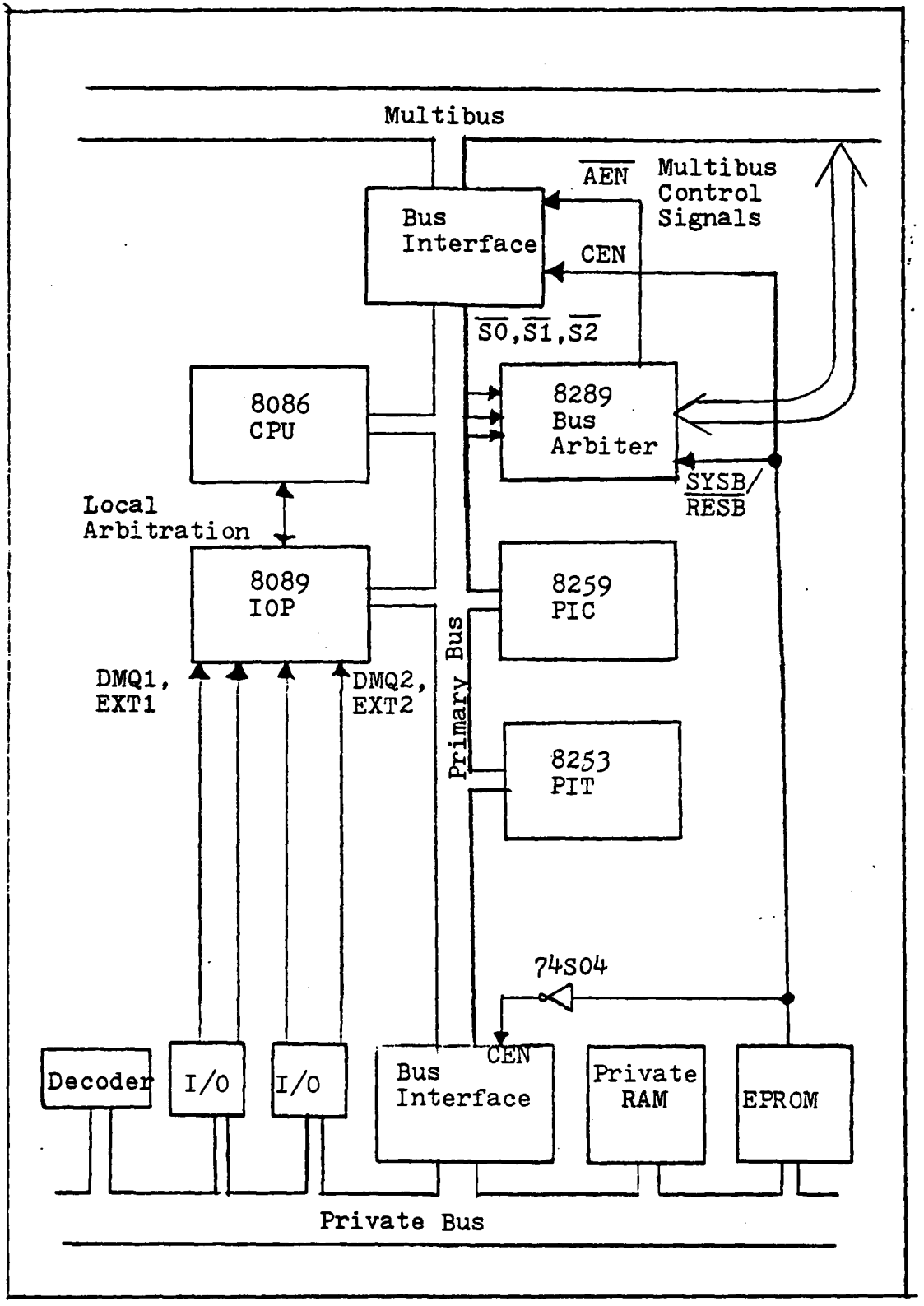


Figure 3-5 Network Subsystem

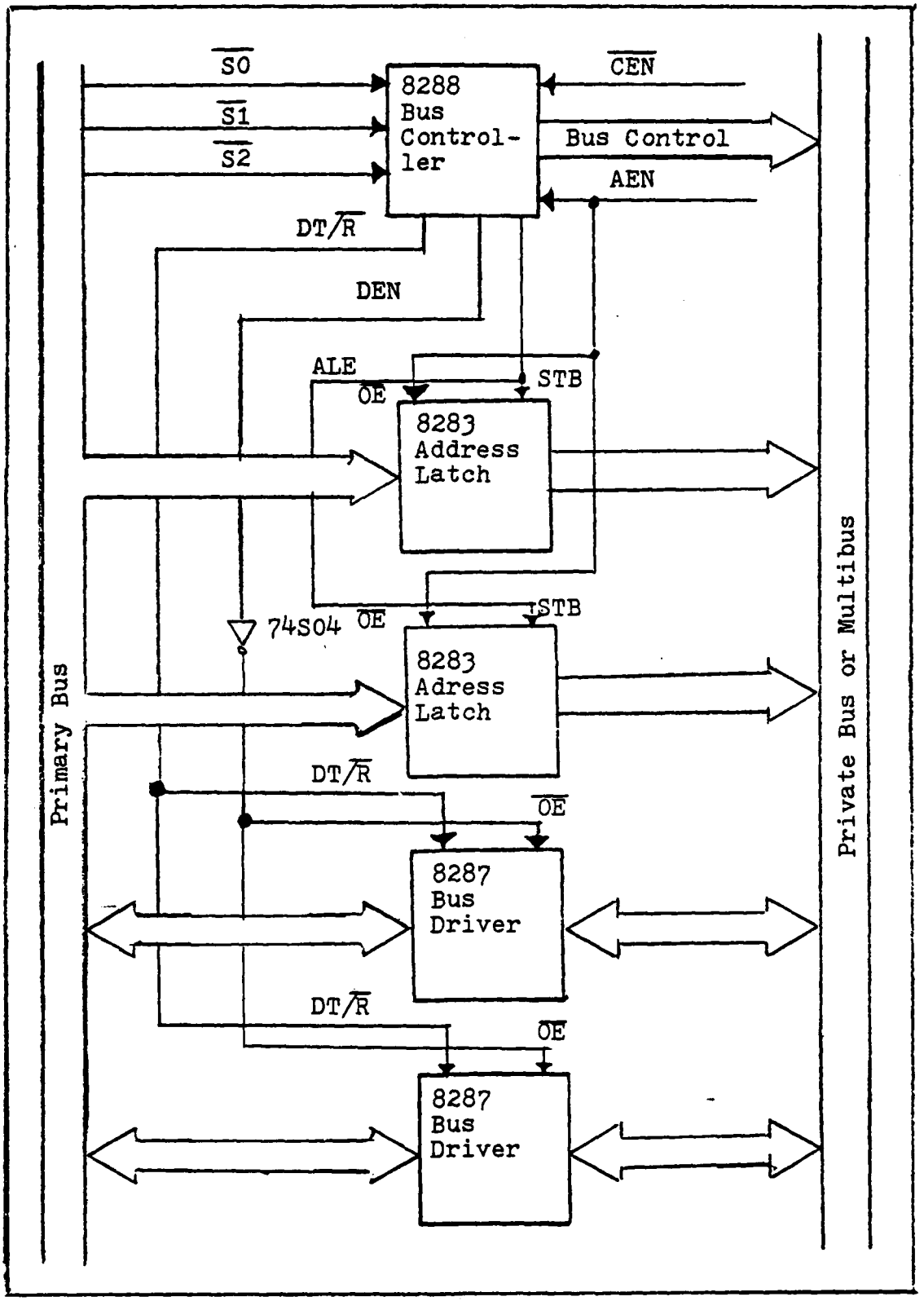


Figure 3-6 Bus Interface Logic

available. UNID II's Network I/O subsystem was designed for prototyping on a Multibus-compatible wire-wrap card. The Network I/O subsystem contains an 8086 CPU, a slave 8089 I/O processor, and private memory and I/O resources. With the exception of supervisory commands and message transfers, the entire Network Subsystem operates in parallel with and independent of the system Multibus.

Figure 3-5 shows the Network Subsystem in more detail. A multiplexed Primary bus is connected to the local processors. Address latches within bus interface logic are used to demultiplex the Primary bus for communication with either the Multibus or Private bus (Figure 3-6). Bus interface logic also contains an 8288 Bus Controller which decodes memory and I/O commands from the processors' status lines (S0,S1,S2) and enables the Primary bus to communicate with either the Multibus or Private bus. (If the Private bus used only I/O commands, the 8288 in the Private bus interface would not be needed. The second bus controller allows the processors to use both memory and I/O commands in the Private address space.) To map some of the local CPU memory space onto the Private bus, an EPROM decoder enables the Command Enable (CEN) input of either bus controller which allows the bus controller to enable its data buffers. Depending on the address, memory instructions will access either the Multibus or Private bus. To allow the EPROM on the private bus to control the CEN input of both bus controllers, the address latches on the Private bus are permanently enabled.

The 8289 Bus Arbiter determines whether to grant a request to use the Multibus and controls the Address Enable (AEN) input of the bus controller attached to the Multibus. The 8289 uses standard Multibus control signals and allows parallel or daisy-chained arbitration between component boards connected to the Multibus (Ref 28:6-18 to 6-20).

The 8089 IOP does not support interrupts in the recognized sense (Ref 28:2-22). However, I/O may be synchronized by using the DMA request input of each channel (Ref 18:4-5). Since each of the two I/O channels is dedicated to a single Network transmission link, no polling is necessary.

To support UNID II's functional requirement for a message timer to keep track of time elapsed before receiving an acknowledgement (Figure 2-6), the Network Subsystem also contains a Programmabe Interval Timer (PIT) and a Programmable Interrupt Controller (PIC). The PIT allows the generation of time delays under software control. After a specified time interval, the PIT will interrupt the local 8086 which can handle the initiation of message retransmission itself or direct the 8089 IOP to the interrupt service routine.

A decoder is used to determine the addresses of the communication ICs and the PIT counters. The decoder may also be used to decode the 8089 IOP Channel Attention and Select signals from an address generated by the local CPU (Ref 18:4-46).

Local I/O Subsystem

The Intel 86/12A Single Board Computer (SBC) was selected to prototype the Local I/O subsystem and shared system memory. This printed circuit card contains an onboard bus for private EPROM and I/O resources, an Intel Multibus interface, and a system bus clock. A dual-port controller allows onboard memory to be used for either shared system memory or private CPU memory. Jumper connections are used to dedicate a portion of onboard memory for exclusive CPU use while the remaining memory is shared system memory. 32K of onboard memory is supplied with the 86/12A SBC but is expandable to 64K with the aid of a special expansion board which mounts directly on top of the 86/12A board. Similarly, the 16K EPROM capacity may be expanded to 32K.

The 8086 SBC contains one RS-232-C serial interface and 24 programmable I/O lines which, by themselves, are not sufficient to handle UNID II's peripheral requirements. Figure 3-4 shows a 4-channel serial I/O board which plugs directly into the system Multibus. The capability to plug commercially-obtained I/O interfaces directly into the Multibus gives UNID II the capability to quickly and easily alter or add local I/O interfaces for different applications. If more I/O interfaces are required for a specific application, another I/O board may be plugged into the Multibus. Some applications may require a parallel interface card while situations with heavy throughput and many I/O interfaces would require another independent I/O subsystem

containing another processor and private memory resources rather than the slave 4-channel board. When UNID II is used as a gateway or network-to-network interface, two high-speed Network I/O subsystems are required instead of one Local and one Network subsystem.

CPU-8089 Communication Protocol (Figure 3-7)

The CPU and 8089 I/O processor (IOP) communicate through a block of shared memory. Figure 3-7 illustrates the communication scheme which uses control blocks connected in a linked structure (Ref 12:50). This section applies standard CPU-IOP protocol to UNID II's Network side which contains the IOP. Standard Intel terminology is delimited by quotation marks.

Each IOP has two independent channels and two independent sets of registers. Since the two channels share the same internal bus, only one channel is active at a time. The IOP may be programmed to give one channel a higher priority than the other or to allow equal priority between its two channels. If the two channels are programmed to interleave bus cycles, applications programs can treat the IOP as two simultaneously operating and independent I/O channels. Figure 3-7 shows that the "channel control block" in shared memory allows the CPU to communicate with either of the IOP's channels. Through the "channel control word", the CPU may stop, start, or suspend either channel. The CPU prepares the "channel control word" and pulses the IOP's "channel

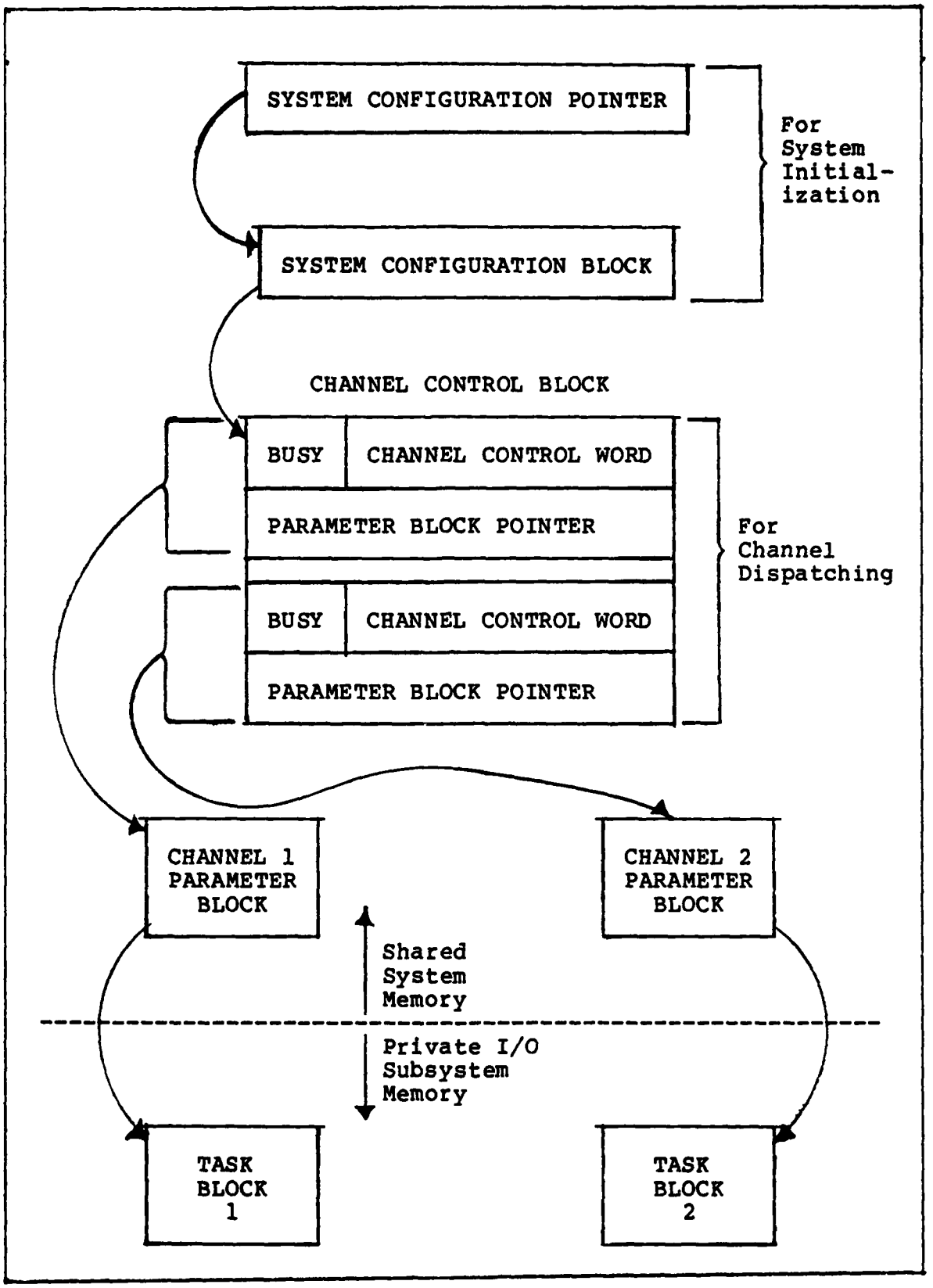


Figure 3-7 CPU-IOP Communication Protocol (Ref 18:3-3)

attention" (CA) input pin which is similar to a CPU interrupt. The state of the IOP's channel select (SEL) input directs the CA to either of the two channels (Ref 18:3-4). After a channel has completed its task, it signals the CPU by setting its busy/done flag in the "channel control block" or activating its interrupt request output, SINTR1 (SINTR2 for channel 2) to send an interrupt request to the CPU.

I/O device driver programs (object code) are stored in private IOP memory in a "task block" while data and variables are stored in a relocatable "parameter block" located within shared memory. The parameter block contains a pointer to the entry point in the "task block".

After an 8089 reset, the 8086 signals the 8089 CA input which causes the 8089 to execute its internal ROM initialization program. During the first CA following a reset, the state of the SEL input determines whether the 8089 will act as a master or slave. For UNID II, the CPU acts as a master and normally has bus control while the slave 8089 recieves bus control only on request. The 8089 IOP reads the "system configuration pointer" to determine the physical width of the system data bus and the beginning address of the "system configuration block". This block determines the private bus width (8 or 16 bits), and defines shared-bus protocol between master and slave processors. The set of conditions under which transfer of local bus control occurs is specified by bit 1 of the "system operation command" (SOC) byte

contained in the "system configuration block". If bit 1 is set to 1, bus transfers occur only if the IOP is idle on both channels; but if bit 1 is set to 0, bus transfers occur when the IOP is idle, waiting for a DMA transfer, or has just completed execution of an unchained instruction (Ref 28:3-2). For a hardware configuration similar to UNID II's where the 8089 IOP is used as a slave to a local 8086, Intel literature states that bit 1 of the SOC byte must be 0 (Ref 18:3-35). After the 8089 has been intialized, subsequent CA signals issued by the CPU are used to start I/O operations in either channel (depending on the level of the 8089 SEL input). Prior to initiation of I/O operations, the CPU prepares the "channel control block" and loads the "parameter blocks".

Each I/O channel both executes programs and performs DMA-like data transfers. (8089 DMA transfers are discussed in Appendix B). Since only one channel may be active at any time instant, DMA transfers normally have priority over channel program execution which could cause a very long delay to one channel's program execution if the other channel is performing DMA transfers. If desired, one channel's program execution can be "chained" so that it has the same priority as the other channel's DMA transfers. Now, IOP hardware alternately allocates bus cycles to each independent channel. This interleaving of bus cycles causes no additional time delay or overhead.

Intel 86/12A SBC Memory Addressing

Following a reset, both the 8089 IOP and 8086 jump to the same area of shared system memory. The 8089 reads the contents of the System Configuration Pointer located at location FFFF6H, and the 8086 jumps to EPROM location FFFF0H which contains a jump instruction to the start of the 8086 program.

For the UNID II prototype, shared memory resides on the 86/12A Single Board Computer. The upper memory that the 8089 (and second 8086) in the Network subsystem must access upon a reset is reserved for the 86/12A monitor and cannot be addressed from the shared system bus. To resolve this conflict, 86/12A shared memory (locations 0 to 7FFFH) is configured to appear as upper memory (locations F8000H to FFFFFH) to the Network Subsystem (and other Multibus boards connected to the system bus) (Ref 21:3; 45). Using this offset, the 86/12A CPU sets up the 8089 System Configuration Pointer at memory location 7FF6H which corresponds to the system location FFFF6H that the 8089 will jump to upon a reset. Similarly, the 86/12A address 7FF0H which corresponds to system address FFFF0H is initialized with a jump instruction for the 8086 in the Network Subsystem. Figure 3-8 is a memory map for UNID II which and shows both 86/12A addresses and the address offset for Multibus access.

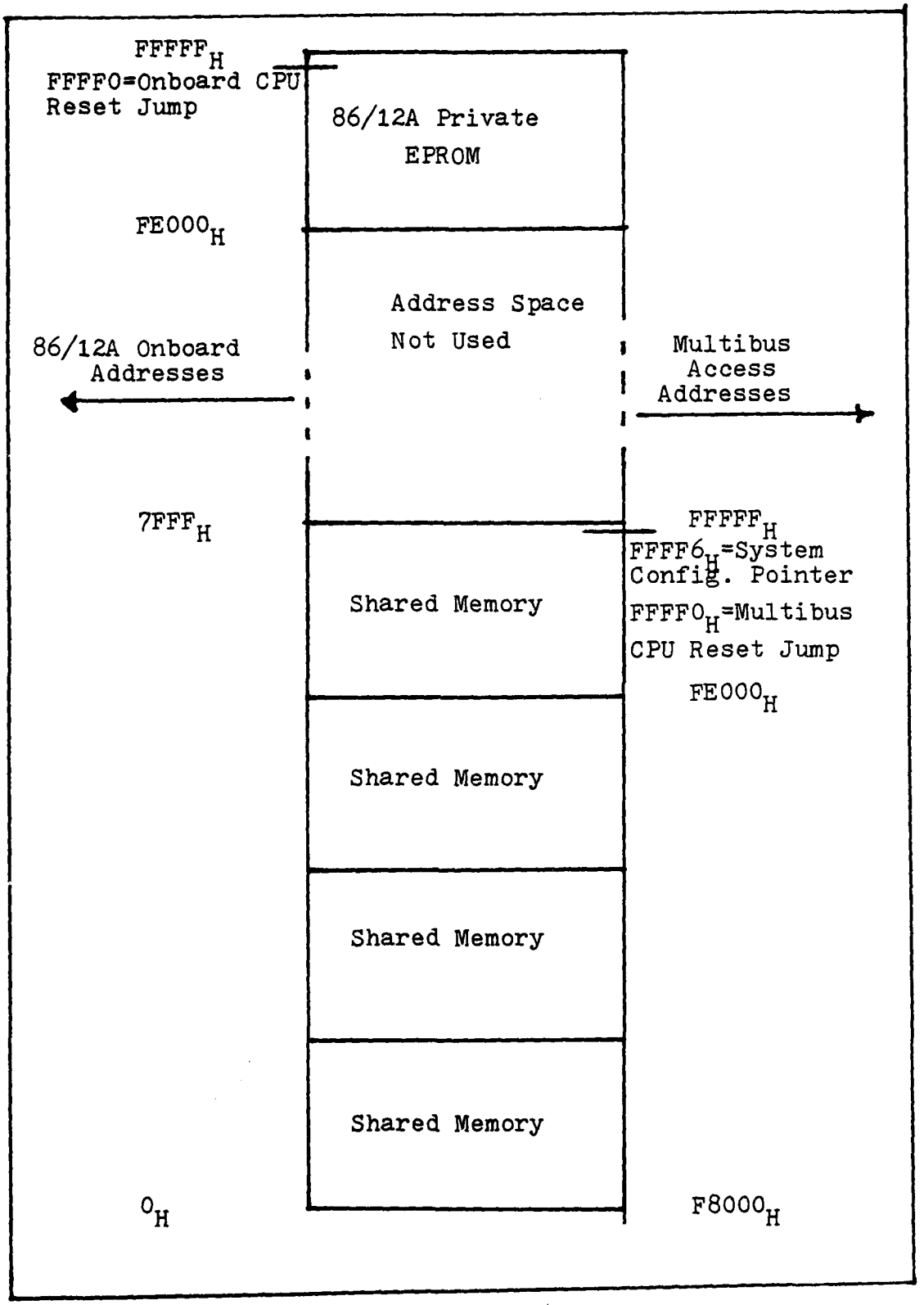


Figure 3-8 UNID II Memory Map

Special I/O Hardware

I/O functions such as parallel-to-serial data conversion or maintaining bit and character synchronization are handled most efficiently with a specialized integrated circuit such as an Universal Synchronous/Asynchronous Reciever/Transmitter (USART) or a multiprotocol chip. Without these specialized communication ICs, processor software would have to transmit data bits at precisely the right time which would monopolize the processor during transmission of each bit.

On UNID II's Local side, specialized I/O hardware is contained on the 4-channel serial I/O board which plugs into the system Multibus. (Some applications may instead require a board containing both serial and parallel or other interfaces). Most of UNID II's lower-speed Local peripherals will use a standard USART to handle specialized communicaton functions.

UNID II's high-speed Network side is likely to use a protocol similar or identical to one of the major link protocols shown in Table IV. Standard USARTs are not well suited for these sophisticated protocols and a multiprotocol IC should be used. The Signetics 2652 was chosen in preference to other multiprotocol chips for three major reasons; (1) the 2652 supports all the protocols listed in Table IV (2) both 8 and 16 bit word lengths can be accommodated, and (3) the 2652 supports transmission rates up to 2

Megabytes/Sec. Table V lists several features of the multiprotocol chips considered while a much more exhaustive list of features is presented in the June 8, 1978 issue of Electronics magazine (Ref 39:112).

Company or Standards Organization	Protocol
IBM	Binary Synchronous Communications (BISYNC)
IBM	Synchronous Data Link Control (SDLC)
ANSI	Advanced Data Communication Control Procedure (ADCCP)
ISO	High-Level Data Link Control (HDLC)
DEC	Digital Data Communications Message Protocol (DDCMP)

Table IV Popular Link Protocols

Number of I/O Ports

Depending on the network topoloy and number of peripherals connected to each UNID, the number of I/O ports required will vary. To allow maximum flexibility, UNID II allows an additional I/O board to be plugged into the system Multibus. This section determines the number of I/O ports that UNID II's basic design can easily accommodate without resorting to additional hardware.

Feature	Sig-netics 2652	SMC 5025	Zilog SIO	Fairchild 3846
Maximum Data Rate (Bits/Sec)	1M/2M	500K	550K/ 880K	1M
Data Bus Pins	8/16	8/16	8	8/16
Multiprotocol (Bisync, DDCMP)	Yes	Yes	Yes	Yes

Feature	Motorola 6854	Intel 8273	Western Digital 1933
Maximum Data Rate (Bits/Sec)	660K/1M	64K	1M
Data Bus Pins	8	8	8
Multiprotocol (Bisync, DDCMP)	No	No	No

Table V Comparison of Communication ICs

Local I/O Subsystem

On UNID II's Local side, the basic number of I/O ports is the number of ports on the commercial I/O board which plugs into the system Multibus. Four ports on a single board is common, but the number may vary. (The single serial port and I/O ports present on the 86/12A SBC are also available for Local peripherals). Intel research (Ref 19:1-12) has determined that an 8085A CPU with 40% of its time devoted to user programs and 60% of its time devoted to I/O can handle 4 simultaneously operating 12,000 baud full duplex channels. (9600 baud would be the maximum standard rate). Since the 8086 has a higher performance than the 8085A, the Local 8086 CPU should be able to comfortably handle the number of I/O channels on the Local I/O board. Data-communications are often "bursty", and it is unlikely that all Local I/O ports will simultaneously require service.

Network I/O Subsystem

Although the 8089 IOP has an advertised DMA transfer rate of 1.25 Megabytes/Sec (0.625 Megabytes/Sec for 8-bit bus widths), UNID II is designed for only two high-speed ports in the Network I/O subsystem. (DMA transfers must be preceded by execution of a channel program whose instruction speeds range from 1.4 to 12.2 microseconds). Each port is dedicated to an I/O channel which greatly simplifies applications programs. After initialization, on-chip logic transparently handles any required time-multiplexing between I/O channels. If more high-speed ports are desired in the Net-

work Subsystem, application programs must poll the various ports.

Communications Interface

Line driver and receiver ICs will be used to allow the actual transmitted signal to meet RS-232-C, RS-422, or RS-423 electrical specifications. Technically, all new devices procured by federal agencies should meet the RS-449 standard (Ref 13:72) which includes both RS-422/RS-423 electrical specifications and the mechanical and functional characteristics of the interface between data terminal equipment (DTE) and data circuit terminating equipment (DCE) (Ref 24:409). Some of the limitations of the older RS-232-C interface are an upper data rate of only 20 Kbits/Sec and a maximum cable length of about 15 meters. The newer RS-423-A standard has a transmission capability of up to 100 Kbits/Sec and is operable with both RS-232-C and RS-422-A. The RS-422-A standard allows data rates up to 2 Mbits/Sec but is not compatible with RS-232-C devices.

Despite the greatly enhanced performance of the newer standards, many manufacturers have been reluctant to change, and most lower-speed peripherals are likely to use RS-232-C. It seemed most benificial for UNID II's lower-speed Local links to use the RS-232-C interface. Even though RS-232-C is operable with RS-423-A, the newer standard uses a larger connector due to an increased number of functional circuits. If all lower-speed peripherals interfaced by UNID II use the RS-232-C interface, the 37-to-25 pin mechanical adapter

required for RS-232-C and RS-423-A interoperability would be an unnecessary inconvenience. UNID II's high-speed Network links will use the maximum performance RS-422-A standard.

Early data-communication peripherals were often electromechanical devices which used current pulses to transmit data. Due to the large number of Teletype machines produced, they are still a dominant type of user terminal (Ref 37:254); and for longer distances, current loops provide greater noise immunity than the popular RS-232-C interface. Unfortunately, circuit resistance, capacitance, and inductance create a lag in the rise and fall of current pulses which greatly limits transmission rates. Current loops tend to create noise in nearby circuits and should be used with caution. Since newer equipment usually uses a voltage interface, UNID II does not directly support a current-loop interface. Commercial RS-232-C to current-loop adaptors may be used if desired. (On UNID II's local side, the iSBC 530 Teletypewriter Adapter may be used to provide an optically isolated 20 mA current-loop interface for the 86/12A board) (Ref 18:B-174).

Similarly, the common-bus type of network architecture is not directly supported. In a bus architecture (Figure 1-7), all messages are simultaneously broadcast to all nodes. If a transciever is designed for the connection between UNID II and the common transmission channel, UNID II may be used to support bus oriented protocols such as the popular carrier-sense multiple-access/collision-detection (CSMA/CD)

protocols. To support Ethernet, a popular CSMA/CD local network, Intel has released a single-board Ethernet controller (iSBC 550) (Ref 14). The iSBC 550 is Multibus-compatible and can be used as UNID II's Network Subsystem.

Summary of Design Philosophy

UNID II is partitioned into two subsystems, the Network Subsystem which handles Network transmission links and the Local Subsystem which handles peripheral links. To allow concurrent processing of Network and Local messages, each subsystem possesses its own processor(s) and memory. Inter-subsystem communication and message transfer occur through a block of shared memory.

In the Network Subsystem, the 8089 I/O processor allows high-speed DMA transfers; but to insure independence from other processors on the system bus, the Network Subsystem also has its own CPU. The Network CPU provides general-purpose instructions for message editing and formatting and arithmetic instructions which may be necessary for generating and verifying error-control words. For maximum performance, the Network Subsystem will use the RS-422 communications standard.

The single-processor Local Subsystem has the capability to quickly and easily add or alter its I/O interfaces. For example, parallel and/or serial I/O boards may be plugged into the system Multibus. (Figure 3-9 demonstrates the overall structure of UNID II). To use UNID II as an internetwork interface, the Local Subsystem should be replaced with a second Network Subsystem.

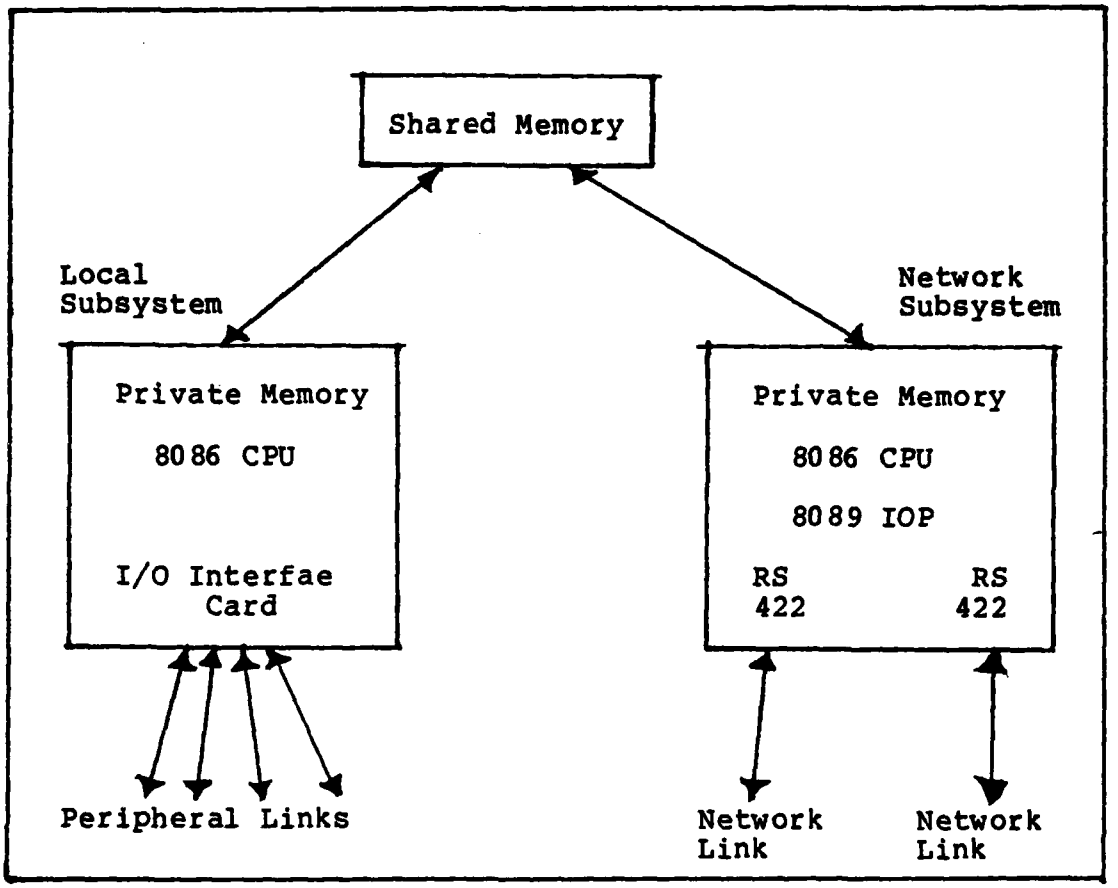


Figure 3-9 Overall Structure of UNID II

IV. Reasons For UNID II's Design

This chapter presents a more detailed description of the reasoning behind UNID II's hardware design which was developed in Chapter III. To illustrate the advantages of the final prototype design, the relative advantages and disadvantages of earlier (and eventually discarded) designs are also discussed. This chapter culminates in the allocation of the functional requirements developed in Chapter II to either the Local or Network subsystem.

Reasons for Using the 8089

Although UNID II's 8089 I/O processor is part of the independent Network subsystem, the 8089 is used as a slave to a local CPU. Since the local CPU is idle whenever the 8089 is using the local bus, it may seem easier to allow the local CPU to execute I/O programs and replace the 8089 IOP with a less complicated DMA controller. This is a viable alternative if it is desired to compromise some of UNID II's flexibility requirements; but to allow maximum flexibility for handling a wide variety of interfacing applications, the 8089 is a better choice. During a DMA transfer (explained in Appendix B), the 8089 can translate network codes and check for end-of-message or other control information using its translate and Mask/Compare options. 8089 on-chip logic transparently handles any necessary word assembly/ disassembly required for using 8-bit peripherals with UNID II's 16-bit data bus. DMA transfers can occur between components (memory or I/O ports) on the private bus, between

components on the system bus, or between system components and private components. The 8089 has two independent I/O channels; and although only one channel may be active at a time, the 8089 supports a multitude of programmer-specified priorities for alloation of bus cycles between channels. If one channel is idle, the other channel operates at maximum capacity even if the idle channel has been initialized and is waiting for information. If both channels are simultaneously active, the interleaving of bus cycles between channels is handled by on-chip logic which does not slow overall throughput. (Depending on channel priorities, one channel's operation may temporarily block the other channel.)

One of the more significant advantages of the 8089 is its ability to execute its own channel programs. This capability modularizes software design. Similar to a mainframe computer, the I/O channels are programmed separetly from CPU programs (Ref 36:301). If a DMA controller is used, the CPU must execute both I/O and CPU programs.

Why the 8089 is not Used in the Remote Mode

In the Remote mode (Ref 28:3-3), the 8089 acts as an independent processor and operates in parallel with the CPU. For example, a CPU used for predominately arithmetic processing can use an 8089 to handle I/O functions and prevent slow-speed I/O from wasting the CPU's time (Figure 4-1). Even though using the 8089 in the Remote mode takes maximum advantage of the 8089's capabilities, the 8089 was

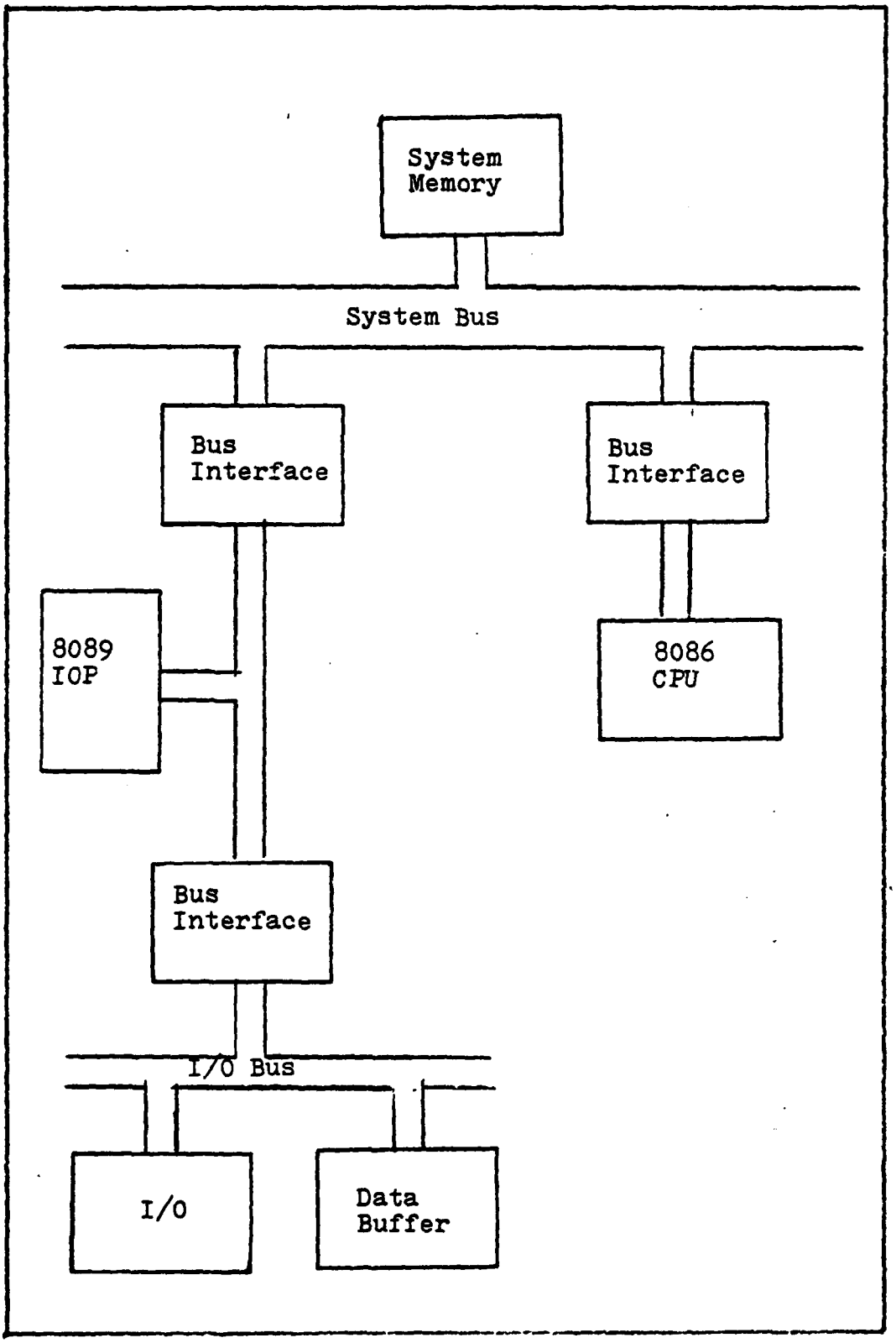


Figure 4-1 Single 8089 In the Remote Mode

used as a slave processor in UNID II's Network Subsystem. The following examples of previous design iterations used the 8089 in the Remote mode and illustrate why an 8089 slave is instead advantageous for UNID II.

Previous Design Iterations

One of the earlier design considerations (Figure 4-2) consisted of two Remote 8089 processors and an area of shared memory for inter-processor communication. Since an 8089 system must contain a CPU to set-up and initiate DMA transfers, an 8086 CPU was included. The problem with this configuration is that it consists of three hardware subsystems while UNID II's Data Flow Diagrams model only two logical subsystems (Network and Local). If one 8089 processes Local messages and the other 8089 processes Network messages, the CPU contributes no processing capability and the added complexity of the CPU interface and arbitration circuitry is not worthwhile. Since the 8089 does not have a lot of general-purpose instructions, this scheme may not be capable of handling UNID II's functional requirements. If the two 8089s are allowed to share the single CPU, bus contention would be unreasonably high. Since both 8089s would require continual use of the system bus during message processing, the advantages of giving each hardware subsystem its own private memory and I/O resources are lost. In comparison, UNID II's final prototype design (described below) contains the same number of processors and less arbitration circuitry than Desin Iteration No. 1 but allows

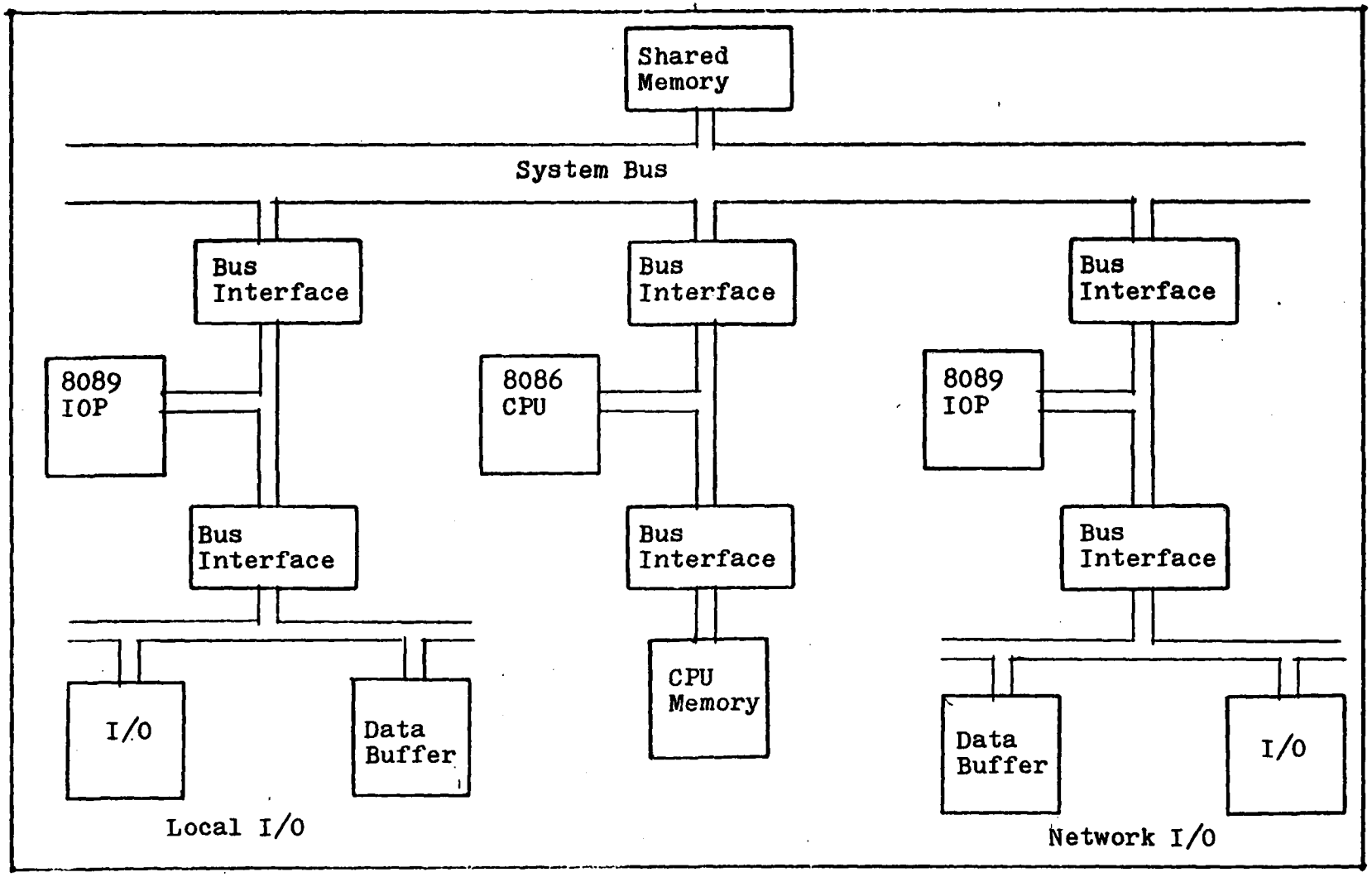


Figure 4-2 Design Iteration No. 1

the Network and Local subsystem to operate relatively independently.

Design Iteration No. 2 (Figure 4-3) is less complex than the previous design iteration, but it has the same problem. The CPU is shared by both subsystems. Even though the two 8089s give the system four independent I/O channels, all channels share the same bus which may be overloaded by high-speed network components.

UNID II's Final Prototype Design

UNID II's final prototype design evolved from Design Iteration No. 2, which has two hardware subsystems. Instead of using a single subsystem for all I/O, Network I/O is handled by one subsystem while Local I/O is handled by the other subsystem (Figure 4-4).

To allow the Network Subsystem to be relatively independent of the system bus, the Network Subsystem contains an 8086 CPU and slave 8089. To allow the flexibiliy of plugging commercial cards into the system bus, the Local Subsystem does use the system bus. Depending on application needs, the commercial I/O card may consist of only I/O ports and USARTs or can consist of an intelligent communications controller board similar to the Intel iSBC 544 (Ref 19: 1-112 to 1-173) which has its own onboard processor and memory in addition to USARTs and buffers. (The final prototype design illustrated in Figure 4-4 is the same design illustrated in Figure 3-4, but Figure 3-4 shows the details of the 86/12A SBC used to prototype the Local Subsystem.)

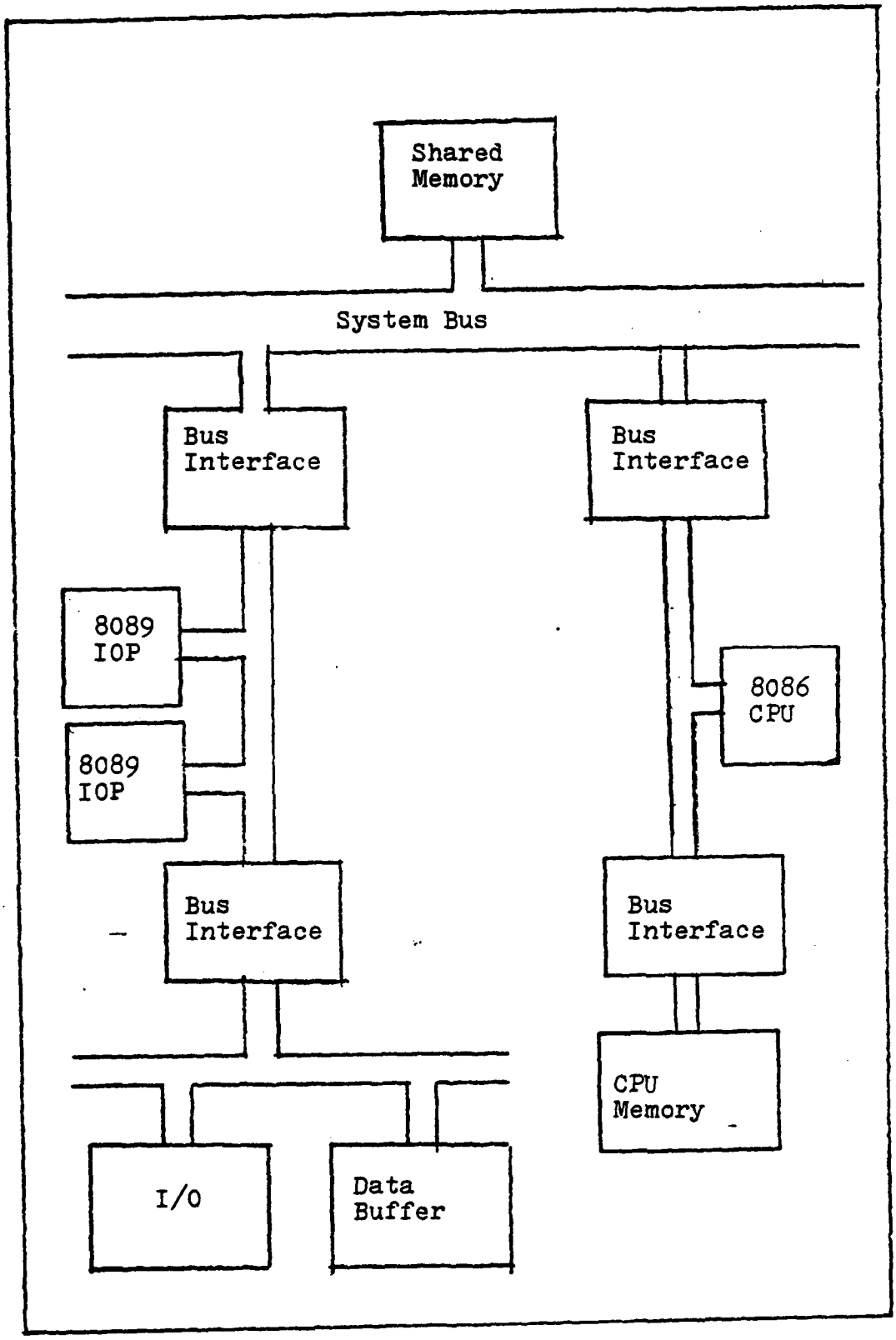


Figure 4-3 Design Iteration No. 2

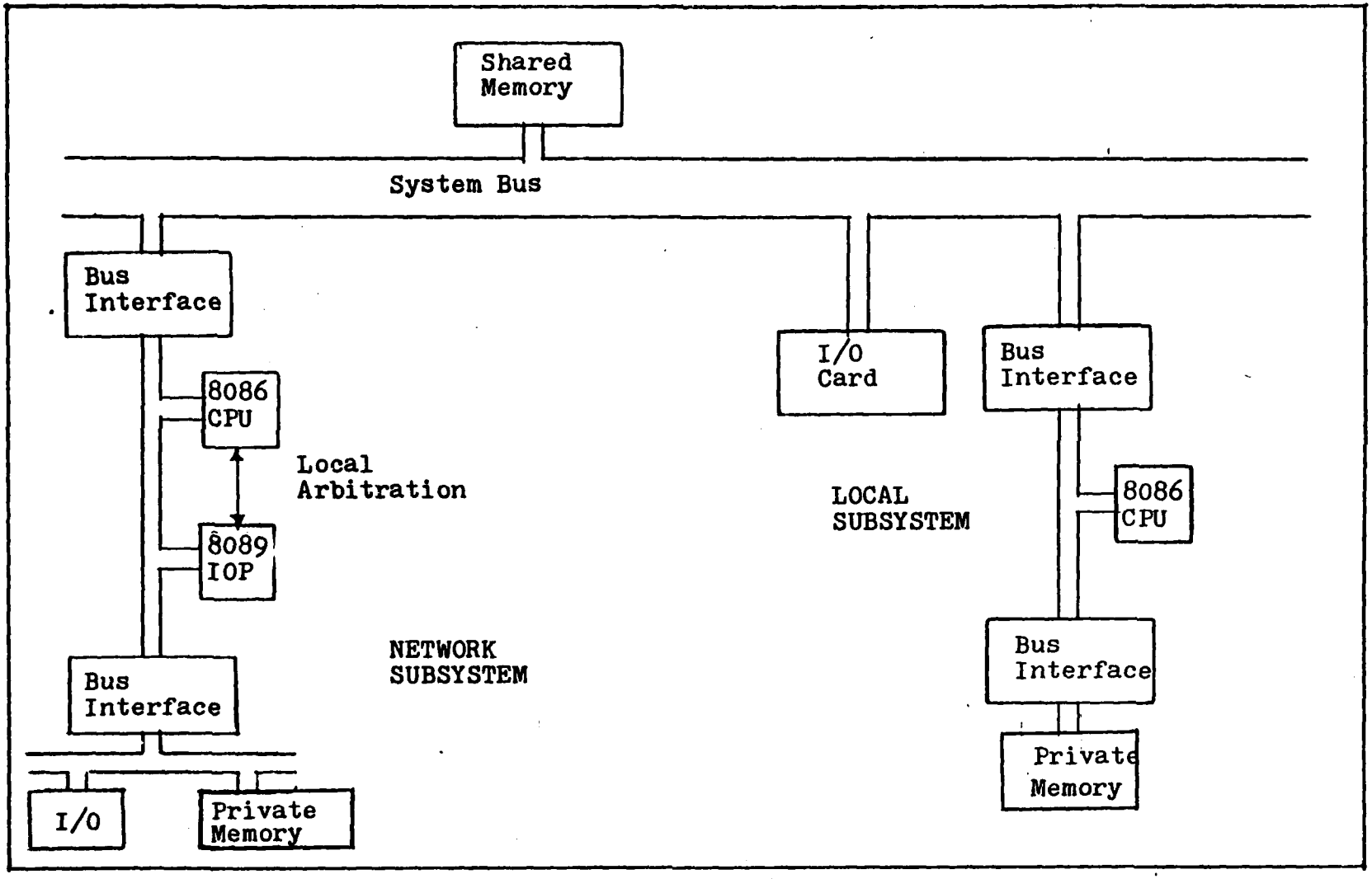


Figure 4-4 Final Prototype Design

Function Allocation

Having determined that UNID II will consist of two hardware subsystems, all Network functional requirements can be allocated to the Network Subsystem and all Local functional requirements can be allocated to the Local Subsystem (Tables VI and VII). (UNID II's functional requirements are developed in Chapter II.) The two following examples illustrate the concept of separating Network and Local functions.

This example consists of a Local terminal communicating with a remote host. UNID II inputs the incoming terminal bit stream into a buffer contained in the Local Subsystem. After a complete message or packet is assembled, it is moved into the shared memory buffer until it can be serviced by the Network Subsystem. The Network Subsystem translates the shared memory packet into the network code and message format as it moves the packet into the Network Subsystem buffer. Once the packet is located within the Network Subsystem, it may be transmitted at a high rate of speed by the 8089 IOP as soon as the appropriate network link(s) are available.

The second example consists of a terminal communicating with a host connected to the same UNID II. In this situation, neither UNID II's Network Subsystem or network links are accessed. (Network links are accessed when local peripherals communicate with remote network components.) The incoming terminal characters are buffered in the Local

Table VI

Local Subsystem Functions

1. Input Local Information
   - 1.1 Check for Transmission Errors
   - 1.2 Transform Serial Data into Bytes or Words
   - 1.3 Store Information in Input Buffer
   - 1.4 Move Information Into Processing Queue
   - 1.5 Correct Input Errors
   - 1.6 Recognize Control Information
   - 1.7 Append Source/Destination Information

2. Format According to Outgoing Protocol
   - 2.1 Assemble Frames
   - 2.2 Translate Code if Necessary
   - 2.3 Identify Routing Information
   - 2.4 Determine Next Destination
   - 2.5 Format Message According to Outgoing Protocol

5. Transmit Local Message
   - 5.1 Transmit According to Local Protocol
   - 5.2 Transform Bytes or Words into Serial Data
   - 5.3 Generate Error Control Information
   - 5.4 Move Message to Output Buffer
   - 5.5 Determine if Local Peripheral is Available

Note: The Above Numbers Correspond to the Data Flow Diagram Nodes Developed in Chapter II

Table VII

Network Subsystem Functions

2. Format According to Outgoing Protocol

   2.1 Assemble Frames

   2.2 Translate Code if Necessary

   2.3 Identify Routing Information

   2.4 Determine Next Destination

   2.5 Format Message According to Outgoing Protocol

3. Transmit Network Message

   3.1 Move Message to Output Buffer

   3.2 Generate Error Control Information

   3.3 Determine if Outgoing Link is Available

   3.4 Transform Bytes or Words Into Serial Information

   3.5 Deallocate Output Buffer Space

   3.6 Set Message Timer

   3.7 Transmit According to Outgoing Protocol

4. Input Network Information

   4.1 Move into Processing Queue

   4.2 Store Information in Input Buffer

   4.3 Transform Serial Data into Bytes or Words

   4.4 Check for Transmission Errors

   4.5 Recognize Control Information

Note: The Above Numbers Correspond to the Data Flow Diagram Nodes Developed in Chapter II

Subsystem memory where they are assembled into a format acceptable to the host. If the Local Subsystem buffer becomes full, some messages can be temporarily stored in shared memory, but shared memory is intended primarily for inter-subsystem message transfer and buffering. When the host link is available, messages buffered locally may be transmitted.

Summary

This chapter began with the reasoning behind the selection of the 8089 rather than a DMA controller. Discussions of previous design iterations demonstrated the value of UNID II's final prototype design which uses one hardware subsystem for Local I/O processing and a second hardware subsystem for Network I/O processing. Two examples describe hypothetical message processing using the constraint that Network messages are processed by the Network Subsystem and Local messages are processed by the Local Subsystem. This constraint serves to minimize UNID II bus contention.

V. Conclusions and Recommendations

This investigation designed a suitable hardware configuration for a programmable network interface. The design was configured to allow maximum flexibility and to accommodate the functional requirements developed in Chapter II. UNID II's flexibility allows it to be used in a wide variety of applications and helps to eliminate the lag time required to custom design hardware for each new application or requirements change to an existing application.

UNID II's design was modularized by allocating Network and Local functions to separate subsystems. The specialized 8089 I/O processor allows the Network Subsystem to transmit and receive data at DMA rates. The 8089's ability to execute its own I/O (channel) programs allows modularization of Network Subsystem software. The Local Subsystem's interchangeable I/O card allows lower-level I/O hardware such as USARTs or I/O ports and line drivers to be easily interchanged.

Recommendations

This investigation indicates that a flexible network interface is a viable project, and the recommendation is to implement the UNID II prototype. The large size of the project suggests implementing UNID II in several stages which are described below.

1) The first step is to write or purchase software to download programs developed on a microcomputer development system to the 86/12A. The 86/12A contains

UNID II's shared memory and forms the basis of the Local Subsystem. Once UNID II software is downloaded to shared memory, UNID II's processors can download appropriate software to private memory areas.

2) The Network Subsystem containing the 8089 can be developed and tested. A complete chip-level schematic and chip-layout diagram is described in Intel Application Note AP-89 (Ref 21) for an 8089 system without private memory or onboard 8086. Similar to UNID II's Network Subsystem, the sample design is intended for wire-wrapping on a Multibus-compatible wire-wrap card. A sample 8089 program and debugging flowchart are also included. Since this example schematic solves a lot of the timing and current-loading problems that occur in a chip-level design, it can be used as the basis for UNID II's Network Subsystem chip-level design.

3) By first operating UNID II using only shared memory, both 8086 and 8089 programs can be examined with the aid of the 86/12A monitor. Private 8089 (Network Subsystem) memory can be designed as an expansion capability.

4) When writing application programs, it is suggested that PLM-86 be used in preference to ASM-86. PLM-86 is a high-level language similar to Pascal. Under some circumstances, the 8086's memory segmentation scheme

tends to make 8086 assembly language confusing. For example, the offset the 8086 uses to address its one megabyte of memory (Ref 18:2-10) can be confusing when assigning an absolute address to an I/O port.

5) After applications programs have been developed and tested, EPROMs can be used to make UNID II independent of a microcomputer development system.

The 8086 family of microprocessors can be quite complex. It is hoped that, in addition to designing a flexible network interface, this investigation served to summarize 8086 multiprocessing circuitry and protocols.

Bibliography

1. Abrams, M.D. "Network Hardware Components," Computer Networks: A Tutorial. IEEE Computer Society, October 1975.

2. Allan, Roger. "Local-Net Architecture, Protocol Issues Heating Up," Electronic Design 29: 91-102 (April 16, 1981).

3. Baker, Lee R. "Prototype and Software Development For Universal Network Interface Device." Master's Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1980.

4. Booth, Grayce M. "Hierarchical Configurations For Distributed Processing," Compcon 77, Fifteenth IEEE Computer Society International Conference, 1977.

5. Brown, Eric F. "Prototype Universal Network Interface Device," Master's Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1979.

6. Carson, John H. "System Design," Design of Microprocessor Systems, edited by John H. Carson. IEEE Computer Society, 1979.

7. Cravis, Howard. "Local Networks Prove Practical for Data Communication Systems in Close Proximity," Datamation: (March 10, 1981).

8. Doll, Dixon, R. Data Communications Facilities, Networks and Systems Design. New York: John Wiley and Sons, Inc., 1978.

9. -----. "Computer-Networking--the Giant Step in Data Communications," Basics of Data Communications, edited by Harry R. Karp. New York: McGraw-Hill Publications Co., 1976.

10. -----. "Multiplexing and Concentration," Computer Networks: A Tutorial. IEEE Computer Society, October 1975.

11. Durniak, Anthony. "Special Report New Networks Tie Down Distributed Processing Concepts," Electronics: 107-122 (December 7, 1978).

12. El-Ayat, K. A. "The Intel 8089: An Integrated I/O Processor," Computer, (June 1979).

13. Folts, Harold C. "A Powerful Standby Replaces the Old Interface Standby," Data Communications: 61-68 (May 1980).

14. Harakal, Joseph P. "Putting Ethernet Onboard the Multibus," Computer Design: 183-186 (September 1981).

15. Hindin, Harvy J. Introduction to "Local Network Gives New Flexibility to Distributed Processing," Electronics: 114-115 (September 25, 1980).

16. -----. "Controlling Data Communications: Statistical Multiplexers Move In," Electronics: 141-148 (July 1981).

17. Hobart, William C. "Design of a Local Computer Network For the Air Force Institute of Technology Digital Engineering Laboratory," Master's Thesis. Wright-Patterson AFB, Ohio. Air Force Institute of Technology, March 1981.

18. Intel Corporation. The 8086 Family User's Manual. October 1979.

19. -----. ISBC Applications Manual. OEM Microcomputers Systems Applications Engineering, Hillsboro, Oregon, 1979.

20. -----. iAPX 88 Book. July 1981.

21. Jigour, Robin. "Prototyping With the 8089 I/O Processor," Intel Application Note AP-89, May 1980.

22. Kahn,Robert. "Terminal Access to the ARPA Computer Network," Computer Networks, edited by Randall Rustin. Englewood Cliffs: Prentice-Hall, Inc., 1972.

23. Kane,Jerry. An Introduction to Microcomputers Volume 3 Some Real Support Devices. Adam Osborne and Associates, Inc., 1978.

24. Krutz, Ronald L. Microprocessors and Logic Design. New York: John Wiley and Sons, 1980.

25. Lorin, Harold. Aspects of Distributed Computer Systems. New York: John Wiley and Sons, Inc., 1980.

26. Morse, Stephen P. The 8086 Primer. Rochelle Park, New Jersy: Hayden Book Company, Inc., 1980.

27. Newport, C.B. and Jan Ryzlak. "Communication Processors," Computer Networks: A Tutorial. IEEE Computer Society, October 1975.

28. Osborne, Adam. 8089 I/O Processor Handbook. Berkely: Osborne/McGraw-Hill, 1980.

29. -----. An Introduction to Microprocessors Volume 1 (Second Edition). Berkeley: Osborne/McGraw-Hill, 1980.

30. Pierce, John R. "How Far Can Data Loops Go?" Computer Communications, edited by Paul E. Green and Robert W. Lucky. New York: IEEE Press, 1975.

31. Ravenscroft, Donald L. "Electrical Engineering Digital Design Laboratory Communications Network," Master's Thesis. Wright-Patterson AFB, Ohio. Air Force Institute of Technology, December 1978.

32. Rector, Russel and George Alexy. The 8086 Book. Berkely: Osborne/McGraw-Hill, 1980.

33. Schwartz, Mischa. Computer-Communication Network Design and Analysis. Englewood Cliffs: Prentice-Hall, Inc., 1977.

34. Sluzevich, Sam C. "Preliminary Design of A Universal Network Interface Device," Master's Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, December 1978.

35. Spronson, C.J. van and D.A. Bearzatto. "Data Communication Protocol Processor," Microcomputer Architectures, edited by Jean-Daniel Nicoud et al. Amsterdam: North-Holland Publishing Company, 1977.

36. Stamm, Dave. "Free the uC's CPU from I/O Hassels With a Special I/O Processor," Tutorial: Microprocessor System Design and Techniques, edited by Carol Anne Ogdin. New York: IEEE, Inc., 1980.

37. Wang, Kuei-Seng and Stephen A. Dalyi. "Don't Overlook Electrical Compatibility When Mating Equipment," Basics of Data Communications, edited by Harry R. Karp. New York: McGraw-Hill Publications Company, 1976.

38. Weinberg, Victor. Structured Analysis. New York: Yourdon Press, 1979.

39. Weissberger, Alan J. "Data-Link Control Chips: Bringing Order to Data Protocols," Electronics: 104-112 (June 8, 1978).

40. Wood, David C., et al. "A Cable-Bus Protocol Architecture," 1979 Data Communications, Sixth Data Communications Symposium. IEEE, Inc., 1979.

41. Intel Corporation. "MCS-86 Software Development Utilities Operating Instructions For ISIS-II Users." Manual Order No. 9800639B (1978).

42. _____. "ISIS-II PL/M-86 Compiler Operator's Manual." Manual Order No. 9800478A.

43. _____. "Universal PROM Programmer User's Manual." Manual Order No. 9800819-01 (1978).

44. _____. "PL/M-86 Programming Manual." Manual Order No. 9800466A.

45. _____. "iSBC 86/12A Single Board Computer Hardware Reference Manual." Manual Order No. 9800645.

Appendix A

The 8086/8089 Software Development Process (Refs 41, 42, 18:3-63 to 3-86)

Appendix A outlines the software development process for an 8086/8089 system using Intel development software which runs on an Intellec 800 or Series II Microcomputer Development System. Although the 8089 I/O processor requires its own assembly language (ASM-89) and assembler which is separate from the 8086 assembly language (ASM-86) and assembler, the 8086 Link and Locate programs operate on segments from both 8089 and 8086 source-code translations (object files). PLM-86 is is a high-order language which is very similar to Pascal. Figure A-1 briefly outlines the 8086/8089 software development steps.

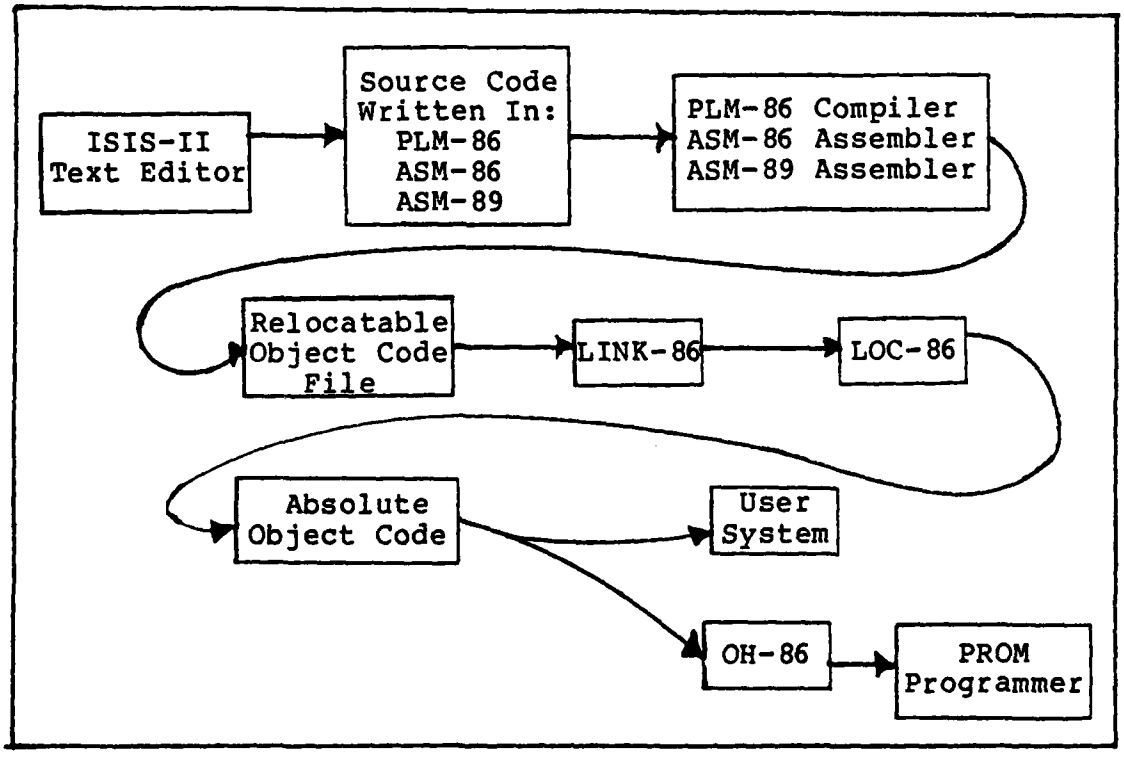


Figure A-1 8086/8089 Software Development

Relocatable Object Code Segments

An object file consisting of relocatable segments results from the use of the appropriate compiler or assembler on a source code module. Segments are the basic elements for linkage and relocation, and segment addresses are relative to the beginning of the segment. (External references may be specifically declared to allow separately compiled modules to reference each other).

Segments are related to the 8086 addressing scheme. Even though the 8086 has a 20-bit (one megabyte) address space, its registers are only 16 bits wide. Four segment registers each which point to a 64K block of memory are used for memory addressing. If more than 256K (4 X 64K) of memory is needed, the program must alter the values of the segment registers. A 20-bit segment address is constructed by appending four lower-order zeros (Ref 26:14) to the 16-bit value in the segment register which allows segments to be placed on any memory boundary that is a multiple of 10 Hex. If desired, segments may be overlapped.

Programs are divided into segments according to certain attributes. For example, a program's object code, stack space, and data are often placed in different segments. ASM-86 allows the programmer to name segments while ASM-89 produces a single logical segment and PLM-86 generates its own segment names.

PLM-86 Object File Sections

Table VIII demonstrates how PLM-86 generates segment, class and group names depending on the compiler option SMALL, MEDIUM, or LARGE (described below). (Experience has shown that the PLM-86 Version 2.1 compiler generates the segment names modname_CODE and modname_DATA rather than modname.CODE and modname.DATA as specified in the "MCS-86 Software Development Utilities Operating Instructions For ISIS-II Users" manual (Ref 41:C-1). These names must be correctly specified when the program is being located or LOC-86 will be unable to find the specified segment(s).

The object file consists of five sections which are combined into memory segments according to the compiler's size control (discussed below). (1) The CODE section contains the program's object code. If the LARGE control is used, the CODE section also contains constants. (2) The CONSTANT section contains variables initialized with the PLM-86 DATA statement, REAL constants, and constant lists. (3) The DATA section consists of program variables. (4) The STACK section is temporary storage used during program execution, but its size is automatically determined by the compiler. (5) The MEMORY section is an area in memory which is referenced by the built-in PLM-86 identifier MEMORY, a BYTE array of unspecified length for uninitiated 8086 storage (Ref 44:12-13).

PLM-86 Size Control

The following compiler options affect the manner that

AD-A115 639 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 17/2
PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFA--ETC(U)
DEC 81 A G GRAVIN

UNCLASSIFIED AFIT/GCS/EE/81D-9 NL

2 of 2

END
DATE FILMED
7-82
DTIC

SMALL compiler option:

Segment Name	Class Name	Group Name
CODE	CODE	CGROUP
CONST	CONST	DGROUP
DATA	DATA	
STACK	STACK	
MEMORY	MEMORY	

MEDIUM compiler option:

Segment Name	Class Name	Group Name
modname_CODE	CODE	none
CONST	CONST	DGROUP
DATA	DATA	
STACK	STACK	
MEMORY	MEMORY	

LARGE compiler option:

Segment Name	Class Name	Group Name
modname_CODE	CODE	none
modname_DATA	DATA	
STACK	STACK	
MEMORY	MEMORY	

Table VIII PLM-86 Generated Segment, Class, and Group Names

(Ref 41:C-1)

different object file sections are combined into contiguous 8086 memory segments. Since memory segments have a maximum size of 64K, the way sections are combined into contiguous segments limits the size of source code modules. (A module is a section of code separately created and translated to an object file.)

SMALL Case

When source-code modules compiled with the SMALL control are linked, the CODE sections from all modules are combined into a single segment. CONSTANT, DATA, STACK, and MEMORY sections from all modules are placed in a separate segment. Since there is only one segment for code and data, the 8086 Code Segment (CS) and Data Segment (DS) registers never need to be updated during program execution and the SMALL control allows the greatest program efficiency. The SMALL control may be used if the total size of all CODE sections is less than 64K and if the total size of all data (CONSTANT, DATA, STACK, and MEMORY) sections is less than 64K. The SMALL case is the default case and must be used when the PLM-86 compiler compiles PLM-80 programs for use on an 8086 system.

MEDIUM Case

The MEDIUM control allocates a separate segment for the CODE section of each compiled module. CONSTANT, DATA, STACK, and MEMORY sections from all modules are placed within a single segment allowing this case to be used if the data sections from all compiled modules fit within a single 64K

segment.

LARGE Case

The PLM-86 LARGE control places CODE and CONSTANT sections from each compiled module into a separate segment, and places the DATA section from each compiled module into a separate segment. STACK sections from all compiled modules are combined into a single segment; and similarly, MEMORY sections from all modules are also combined into a single segment.

LINK-86

LINK-86 combines separately translated source-code modules from ASM-89, ASM-86, and PLM-86 and resolves external references between modules. When first defined, the relocateable segments that form the translated modules do not have a fixed size, and if segments from separately translated modules have identical names, they are combined into the same segment. (Segments with different names remain as separate segments.)

LOC-86

LOC-86 assigns absolute addresses to the object code file. The ADDRESSES control may be used to specify addresses for particular SEGMENTS or CLASSES. For example, to place object code contained in the previously-linked file, 8086LNK.OBJ, into EPROM space beginning at location FE000H, the following series of commands may be used. (The name of the CODE segment contained in the input file, 8086LNK.OBJ is PROTOTYPE.CODE.)

```
LOC86 8086LNK.OBJ TO 8086EPROM.OBJ&
BOOTSTRAP&
ADDRESSES (SEGMENTS(PROTOTYPE_CODE(FE000H)))
OH86 8086EPROM.OBJ TO 8086EPROM.HEX
```

After a power-up or reset, the 8086 jumps to the top of its memory space to location FFFF0H. The BOOTSTRAP control may be used to place a long jump in locations FFFF0H through FFFF4H so that program control will be directed to the beginning of the program code segment. The following LOC-86 memory map would be generated by the above LOC-86 commands:

```
MEMORY MAP OF MODULE PROTOTYPE
READ FROM FILE 8086LNK.OBJ
WRITTEN TO FILE 8086EPROM.OBJ

MODULE START ADDRESS PARAGRAPH = FE00H OFFSET = 00AEH
SEGMENT MAP

START     STOP      LENGTH ALIGN NAME            CLASS

00200H    0032CH    012DH    W   PROTOTYPE_DATA DATA
0032EH    0036FH    0042H    W   STACK          STACK
FE000H    FF6BAH    16BBH    W   PROTOTYPE_CODE CODE
FFFF0H    FFFF4H    0005H    A   (ABSOLUUTE)
FFFF6H    FFFF6H    0000H    W   MEMORY         MEMORY
```

Since no specific address was specified for DATA and STACK segments, they are located by default beginning at location 200H. The above number values were taken from a sample program, but the actual segment lengths are dependent on the source program.

OH-86

Before a located object file can be printed in readable form, it must be converted to HEX by OH-86. Also, the input to an INTEL Universal Prom Programmer (UPP) must be an 8086 Hex file.

Using the Intel Universal PROM Programmer (UPP) for 8086/8089 Development (Ref 43:C-9)

8086 Hex files are stored differently than 8080 Hex files, and the command READ 86HEX FILE must be used. Two parallel 8-bit wide PROMs are used to form 16-bit words; and for the same block of PROM memory, one PROM contains even-address, low-order bytes while a second PROM contains odd-adress, high-order bytes. Since Intel Universal PROM Mapper (UPM) software only allows one PROM at a time to be programmed from a single sequential Hex file, both the high-order and low-order bytes from the input Hex file must be separately collected into contiguous sections of memory. The STRIP command may be used to sequentially read through an 86 Hex file and strip off either high or low order bytes and place the remaining bytes in a new sequential file. Table IX is an example of 7 steps for programming four 2716s (Ref 21:4):

Step	UPP Command
(1)	READ 86HEX FILE 8086EPROM.HEX INTO 2000H
(2)	STRIP LOW FROM 0 TO 1FFFH INTO 4000H
(3)	STRIP HIGH FROM 0 TO 1FFFH INTO 6000H
(4)	PROGRAM FROM 4000H TO 47FFH START 0
(5)	PROGRAM FROM 4800H TO 4FFFH START 0
(6)	PROGRAM FROM 6000H TO 67FFH START 0
(7)	PROGRAM FROM 6800H TO 6FFFH START 0

Table IX UPP Programming of four Intel 2716 EPROMs From an 8086 Hex File

(1) The entire 86 Hex file is read into the memory of the Intellec 800 or Series II. (2) The low-order bytes are stripped away from the input file leaving the high-order bytes which are stored sequentially beginning at location 4000H. (3) Similarly, the high-order bytes are stripped away, and the result is stored starting at location 6000H. (4 & 5) Using the previously created file of high-order bytes, the high-order byte PROMS are programmed. (6 & 7) The low-order byte PROMs are programmed.

Caution for Assembly Language Programmers

Due to the 8086 six-byte instruction queue which prefetches instructions when the bus is not busy, code should not be written within six bytes of physical memory. The 8086 may attempt to prefetch nonexistent code which will hang the processor if the 8086 system is configured to wait for the READY acknowledgement from addressed memory (Ref 18:2-96).

Appendix B

8089 DMA Transfers (Refs 18:4-47 to 4-51; 28)

Since all 8089 DMA transfers require at least two bus cycles, one cycle to read a data byte or word from a source address and one cycle to write the data to a destination address, the 8089 does not provide standard "DMA" which transfers a block of data in a single cycle. However, 8089 DMA transfers are very fast. Using a standard 5 Mhz clock, the 8089 can achieve DMA transfer rates up to 1.25 Megabytes/Sec (Ref 28:1-1).

8089 Channel Program

The 8089 I/O processor both executes I/O programs and performs DMA transfers. All I/O operations, including the transfer of a single byte, are handled as DMA transfers; and to initiate a DMA transfer, a channel program must be executed. The channel program loads the 8089's Channel Control Register which specifies the source and destination addresses, data translation option, logical bus widths, type of data synchronization, and DMA terminate conditions. Additionally, the channel program sets mask and counter registers (discussed below) and starts the DMA transfer with a XFER instruction.

8089 DMA Terminate Conditions

The 8089 can exit the DMA mode and return to the channel program on one or more of the following conditions. (1) A single cycle transfer may be specified so that channel

execution resumes immediately after the transfer of a word or byte. This mode may be desireable if a single 8089 I/O channel is polling several lowspeed devices such as CRTs. (2) The Mask/Compare terminate option allows termination on either a successful or unsuccessful match of the Mask/Compare register. This feature can be used to recognize end-of-message characters in UNID II's incoming bit stream. (3) DMA can be terminated when the Byte Count register decrements to zero. (4) Each of the 8089's two channels has an External Terminate (EXT) input signal to allow external devices to control the DMA transfer.

Code Translation Option

The 8089 allows data to be translated during a DMA transfer. To use this option, a 256 byte translation table is created in memory. The source byte is used to index the table, and the indexed byte is sent to the data destination. Since this option only works for bytes (not words) 8-bit logical bus widths must be specified.

Data Source and Destination Options

During a DMA transfer, the 8089 reads from a source address, latches the data, and transfers the data to a destination address. Both the source and destination address may be directed toward either the 8089's shared or private bus, and either bus may be 8 or 16 bits wide. On-chip logic transparently handles any necessary word assembly/ disassembly. Both the source and destination may be treated as either an I/O port or memory, but memory addresses are automatically incremented after each byte or word transfer

while I/O port addresses remain unchanged.

Data Synchronization

The 8089 allows unsynchronized, source-synchronized, and destination-synchronized DMA. Since unsynchronized DMA transfers allow maximum transfer rates, they are usually used for memory-to-memory transfers. For I/O transfers, UNID II must not send or receive data faster than communication links (and communication ICs) allow. In the synchronized modes, the communications interface (or I/O device controller) initiates each 8089 data transfer cycle. Source synchronization is used for I/O reads while destination synchronization is used for I/O writes. Each of the two 8089 channels has its own DMA Request (DRQ) input. The 8089 acknowledges a DMA Request by placing the I/O port's address on the bus, and external logic must be used to decode this address as a DMA Acknowledge (DACK) to the DMA Request (Ref 18:4-51).

Confusion Between Channel Control Byte and Channel Control Register

The Channel Control Register discussed in this appendix is a register within the 8089 used for specifying DMA options and should not be confused the Channel Control Byte located in the Parameter Block within shared memory. The Channel Control Byte is used by the CPU to stop, start, or suspend an 8089 I/O channel. To avoid confusion, both Intel (Ref 18) and Chapter III of this paper use the term Channel Control Word. Osborne (Ref 28) uses the term Channel Control Byte.

Vita

Andrew G. Gravin was born on October 23, 1955 in Amarillo, Texas. In 1974, he graduated from Foothill Senior High School in Sacramento, California. He attended California State University, Sacramento (CSUS) and received the degree of Bachelor of Science in Electrical and Electronic Engineering in May 1980. As a Senior at CSUS, he was a part-time civilian employee for the 1155 Technical Operations Squadron, McClellan AFB, Sacramento. He participated in a ROTC cross-town program with the University of California at Berkeley and received an Air Force commission in June 1980. He entered the Air Force Institute of Technology in July 1980.

Permanent address: 5250 Elbert Way
Sacramento, California 95842

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/81D-9	2. GOVT ACCESSION NO. AD-A115 63	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PRELIMINARY DESIGN OF A COMPUTER COMMUNICATIONS NETWORK INTERFACE USING INTEL 8086 AND 8089 16-BIT MICROPROCESSORS		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Andrew G. Gravin, 2Lt, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433		12. REPORT DATE December 1981 13. NUMBER OF PAGES 106
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Dean for Research and
Professional Developme t
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

LYNN E. WOLAVER
Dean for Research and
Professional Development

APPROVED FOR PUBLIC RELEASE; IAW AFR 190-17
Frederic C. Lynch, Major, USAF
Director of Public Affairs
24 JUN 1982

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Network Interface
Intel 8086 Microprocessor
Intel 8089 I/O Processor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract

This research describes the design of a Universal Network Interface Device (UNID II) which is intended for use in a computer communications network. The II distinguishes UNID II from the original UNID which was also designed and developed at the Air Force Institute of Technology. UNID II's purpose is to lessen the time delays and development costs incurred by custom-designing network interfaces for each application. UNID II is a programmable interface; and although different applications require different device dependent programming, UNID II hardware remains essentially unchanged. A requirements study shows that to handle a wide variety of interfacing situations, UNID II must perform node functions which include concentration, switching and routing, front-end processing, and user-terminal interfacing. The performance of these functions relieves network hosts from communication-specific software.

The key design concept is the subdivision of UNID II into two independent subsystems which communicate through an area in shared memory. The Network Subsystem handles high-speed network links while the Local Subsystem handles network peripherals. This modular approach reduces bus contention and allows the effect of an error or change to be isolated.

For high performance, the Network Subsystem includes the Intel 8089 I/O processor; and for flexibility, the Local Subsystem uses a Multibus-compatible communications board which contains serial or parallel interfaces and may be interchanged depending on user needs.