NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ANALYSIS AND DESIGN OF A
PARAMETERIZED PROTOCOL CONVERTOR

by

Berle Garris Jr.

December 1985

Thesis Advisor: M. L. Cotton

CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
. DECLASSIFICATION / DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S)	S. MONITORING ORGANIZATION REPORT NUMBER(S)
. NAME OF PERFORMING ORGANIZATION aval Postgraduate chool — 6b. OFFICE SYMBOL (If applicable) 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
ADDRESS (City, State, and ZIP Code) onterey, California 93943-5100	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5100
NAME OF FUNDING / SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
NALYSIS AND DESIGN OF A PARAMETERIZED PROTOCOL CONVERTOR

PERSONAL AUTHOR(S)
erle Garris, Jr.

TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
aster's Thesis	FROM _____ TO _____	1985 December	109

SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Protocol Convertor; Data Flow; VLSI

ABSTRACT (Continue on reverse if necessary and identify by block number) Designed to circumvent the incom-
atibilities between communicating computer systems, a parameterized protocol convertor
ermits the use of communication equipment supporting variations of the same communication
rotocol or completely different framing technique protocols. The analysis of the
onversion process includes the engineering trade-offs between speed of conversion and
lexibility, and the use of an alternative flow architecture. Flexibility is enhanced
hrough user selection of input and output protocol types, and the designation of
unctional specifics, such as code type, header length, and error detection methods, with
ariable parameters. The speed of conversion is increased through the parallel processing
f the framing, transparency, and error control sub-functions and the use of a single byte
torage technique. The single byte storage technique imposes some limitations in the use
f transparent data.

. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
. NAME OF RESPONSIBLE INDIVIDUAL rof M. Cotton	22b. TELEPHONE (Include Area Code) 408-646-2377 — 22c. OFFICE SYMBOL 62Cc

FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

Analysis and Design of a Parameterized Protocol Convertor

by

Berle Garris Jr.
Captain, United States Marine Corps
B.S., United States Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1985

ABSTRACT

Designed to circumvent the incompatibilities between communicating computer systems, a parameterized protocol convertor permits the use of communication equipment supporting variations of the same communication protocol or completely different framing technique protocols. The analysis of the conversion process includes the engineering trade-offs between speed of conversion and flexibility, and the use of an alternative flow architecture. Flexibility is enhanced through user selection of input and output protocol types, and the designation of functional specifics, such as code type, header length, and error detection methods, with variable parameters. The speed of conversion is increased through the parallel processing of the framing, transparency, and error control sub-functions and the use of a single byte storage technique. The single byte storage technique imposes some limitations in the use of transparent data.

TABLE OF CONTENTS

I. INTRODUCTION

Currently there is a proliferation of computers needing to exchange information which are hampered by incompatible communication protocols. These incompatibilities are manifest in different word lengths, different operating speeds, various error detection schemes and assorted other capabilities. Any one of these incompatibilities between two systems effectively renders communication between them impossible.

Capability, not communications compatibility is usually the driving factor in system procurement decisions, and the subsequent communications incompatibility brings on inefficiency. Until communication parameters can be standardized nationally, and then internationally, the need exists for an intermediate solution: a parameterized protocol converter.

This thesis describes an analysis and exploratory design of a parameterized protocol convertor; a protocol convertor with its functional specifics designated with variable parameters. Designed to circumvent the incompatibilities between communicating computer systems, the parameterized protocol convertor permits the use of communication equipment supporting variations of the same communication protocol or completely different framing technique protocols. The parameterized protocol convertor is adaptable to any combination of input and output protocols of the three major framing techniques. The three major framing techniques are character oriented protocols, byte count protocols, and bit oriented protocols.

There are two initial design requirements to be met with a parameterized protocol convertor:

- Sufficient flexibility to absorb variations between implementations of similar protocols.
- The fastest performance possible.

Flexibility is required for several reasons. Various system manufacturers have interpreted the protocol standards differently. In their designs of communication equipment, the system manufacturers have included small but significant variations between the protocol their equipment supports and the protocol other manufacturer's equipment supports. These differences between 'standard' protocols makes communication between systems from different manufacturers difficult.

If the design is to remain viable for any length of time, it must be flexible in its implementation. Changes in the protocols supported and the addition of new protocols are anticipated by parameterizing aspects of the protocols likely to be altered.

Although there are several protocol convertors already available, all suffer from limitations as to their speed of operation or the extent of their flexibility. To increase the speed of conversion, there are several aspects of current protocol convertors which need to be modified. One of these is the storage of data as it is converted. Current protocol convertors must store a large portion of a frame of information to convert the information content between two different protocols. If the data stream can be manipulated without storing the entire frame in the protocol convertor memory, the speed of conversion can be increased.

Another property of current protocol convertors in need of modification is the use of sequential, microprocessor controlled logic to implement the conversion process. Many facets of protocol conversion can be executed simultaneously with parallel processing. There are alternative architectures more adapted to parallel

processing than the traditional control flow architecture employed in current protocol convertors.

Sequential, microprocessor controlled logic usually requires software control of its operation. Software control is desirable for flexibility and ease of alteration of the sequence of operation. But software control also decreases the conversion speed, in that instructions must be fetched and interpreted. The requirement to fetch and interpret instructions can be removed by implementing the protocol conversion algorithms in hardware instead of software.

A synopsis of each chapter follows.

Chapter Two provides some basic definitions of protocols, their sub-functions and protocol convertors. The definitions are followed by a discussion of some of the requirements for protocol conversion. One reason emphasized is a lack of clearly delineated standards in the digital communication field. Another reason is the extensive interoperability requirements of computer communications in both the military and the civilian business world.

Chapter Three is an analysis of the two major conflicting requirements of a protocol convertor; speed and flexibility. A substantial speed of operation is desired to keep the protocol convertor from becoming the bottleneck in the communication system. At the same time, sufficient flexibility is required to absorb variations in the different implementations of similar protocols.

Chapter Four is a description of the parameterized protocol convertor architecture. Starting with an example of how the parameterized protocol convertor would operate, the chapter includes descriptions of the system block diagram, the separate protocol conversion units and several customized circuits. The implementation of the conversion process in hardware using an internal virtual protocol is

described, along with a detailed description of the three sub-functions required for protocol conversion; framing, transparency and error control.

Chapter Five is a description of the implementation of the architecture described in Chapter Four. The implementation description encompasses both the chip design for the parameterized protocol convertor and a system design for the entire communications link including the protocol convertor. Several alternatives for the system design are presented, covering various levels of traffic intensity on the communications channel.

Chapter Six is a summary of the presentation and some conclusions drawn from the analysis and exploratory design.

This thesis presents some possibilities for improving the current state of protocol conversion. Several innovative approaches to the conversion problem are explored and several new techniques developed. The concept of parallel sub-function processing and the concept of the protocol convertor as a filter with minimal storage is supported throughout, even though this leads to some unresolved problems.

II. THE NEED FOR PROTOCOL CONVERTORS

A. DEFINITION OF A PROTOCOL

For two systems to communicate successfully, they must 'speak the same language;' that is they must both understand what is being passed between them as to content, form and timing. The information passed between the two systems must comply with some mutual set of rules and conventions, called a protocol [Ref. 1:p. 1].

The analysis of complex communication protocols and systems can be simplified through the use of partitioning. One set of partitions in communication network theory is the International Standards Organization (ISO) Reference Model of Open Systems Interconnection (OSI), commonly referred to as the seven layer model.

As its common name implies, the ISO OSI model consists of seven layers:

1. The application layer.
2. The presentation layer.
3. The session layer.
4. The transport layer.
5. The network layer.
6. The data link layer.
7. The physical layer.

The bottom two layers, the physical layer and the data link layer are usually implemented in hardware and are the two layers of primary interest in this study of the protocol conversion process.

The physical layer is involved in transmitting raw bits over a communication channel. Here the major considerations are mechanical, electrical, and procedural interfacing to the subnet. The data link layer is involved in segmenting the input data into frames. The data link

layer creates and recognizes frame boundaries by attaching special bit patterns to the beginning and ending of a frame. [Ref. 2:p. 17]

The top five layers are typically implemented in software and perform various tasks such as:

- Controlling the operation of the subnet.
- Determining the route for the frame to follow.
- Providing an interface for the user into the network.
- Executing library functions.

Each communication system in the ISO OSI model consists of an identical set of seven layers. The use of the model leads to analyzing seven different protocols between the seven layers of the model. Within each communication system, messages to be transmitted are passed down through the top layers of the model to the bottom layers of the model through interfaces. These interfaces provide a conduit for data between the layers of the model and serve to insulate the different layers from changes in adjacent layers.

Only the bottom layer of the model, the physical layer, uses a physical protocol. The physical layer is the only layer that actually passes tangible data bits between the two communication systems. The other six layers communicate through implicit protocols. There is no physical link between peer layers of the two communication systems in the top six layers. The passage of data from an upper layer to the bottom layer of one system, across the physical link and back up to the equivalent layer of the other system provides a virtual communication link between the two peer layers.

In addition to the ISO OSI partitioning of the entire communication system, the concept of a protocol can be divided into seven specific sub-functions. According to McNamara [Ref. 3], protocols solve operating problems in the following areas:

- Framing.
- Error control.
- Sequence control.
- Transparency.
- Line control.
- Time-out control.
- Initialization.

The concept of framing or segmenting can be viewed on two separate, yet interconnected levels. Framing can be considered the determination of which groups of bits make up characters, or which groups of characters constitute frames. The current popular protocols are divided into three categories according to their message framing and segmenting techniques. Character oriented protocols use special characters to indicate the beginning and ending of a frame. Byte count protocols send a tally of how many of the characters or bytes following the frame header constitute the information filed of the frame. Bit oriented protocols, like character oriented protocols, use a special flag character or bit sequence to delineate frames.

Byte count protocols are sensitive to undetected errors in the tally field of the frame, and restrict the data format to a specific character size. Character oriented protocols hamper the evolution of the protocol by building in a specific character code. The most popular modern framing technique is the one used in bit oriented protocols. Bit oriented protocols prevent user data from interfering with framing, but do not restrict the data to one particular character size. [Ref. 4:p. 10]

Error control encompasses the entire area of error detection and correction. Various forms of redundancy checks are used to determine if a frame was received without errors. These include, but are not limited to: Cyclic Redundancy Checks (CRC), Longitudinal Redundancy

Checks (LRC) and Vertical Redundancy Checks (VRC). Popular protocols request the re-transmission of error corrupted frames instead of the time consuming process of error correction.

Sequence control is concerned with the numbering of frames to avoid duplication and loss of frames. The re-transmission of error corrupted frames requires sequence control to reduce the possibility of interpreting re-transmitted frames as originals.

Transparency involves transmitting data that could be interpreted as special control characters. Some frames may contain data that appear to the receiving station to be one of the special control characters (for character oriented protocols) or special bit sequences (for bit oriented protocols) used in framing. Bit stuffing and character stuffing are used to alter the data as it leaves the transmitting station and prevent any misinterpretation by the receiving station.

Line or flow control can be viewed as traffic control on the transmission medium. This protocol sub-function determines which station will transmit and which station will receive. The station receiving the frame must reply to the station sending the frame with acknowledgments and possible requests for the re-transmission of error corrupted frames.

Time-out control is that part of a protocol that handles the case of what to do if the message traffic suddenly ceases. It also collaborates with sequence control in keeping track of lost frames by signaling the lack of an expected response after an allotted time period.

Start-up or initialization control handles the case of instigating the flow of data in on idle communication channel. It encompasses the determination of how to inform the receiving station that a frame is on its way before the actual arrival of the frame. This allows the receiving

station to prepare itself to receive the frame.

These are just a sampling of the duties of a protocol. There are other problems that must be solved as well, such as deciding what a transmitter should send when it has no data to send, and how to recover from an abnormal condition. The various protocols solve these problems with a multitude of different methods. Before any effective transfer of information between two different protocols can take place, the different protocol solutions to each of these listed problems must be correlated. This is the challenging job of the protocol convertor.

B. DEFINITION OF A PROTOCOL CONVERTOR

When two communication systems do not use the same protocol, a special type of filter or buffer is needed to support communications between them. The filter accepts data in one protocol and plies it as necessary to transform it into another protocol for output. This data manipulation effectively establishes a data path between the two systems. The established data path permits communication between the two systems despite differences in speed and message formats. The filter operation is called protocol conversion, and consequently the filter has been dubbed a protocol convertor.

The analysis of the protocol conversion process entails many of the functions performed by the protocols themselves. But a distinction must be drawn between the protocol conversion process and the operation of a protocol. Any communication system needing the services of a protocol convertor already has the mechanisms and circuits in place to accomplish the tasks delineated by the sub-functions. These mechanisms and circuits are specific to one particular protocol, but they presumably function properly within the specifications of that particular protocol. The job of the protocol convertor is not that of

a half or full-duplex serial receiver or transmitter, but simply a filter to assist in the transfer of data between two system using different protocols.

Protocol conversion can be accomplished with many different technologies and techniques. Karten [Ref. 5:p. 7] lists several different options for connecting incompatible equipment; each with its own set of advantages and disadvantages.

- A 'black box' hardware approach.
- A software program.
- Network-based protocol conversion services.
- Varied combinations of hardware and software.

The oldest, most established method of protocol conversion is the 'black box' approach; a hardware device connecting two communicating stations. Each station sends its signals to the other station through the 'black box'. Within the box, the received signals are converted into a protocol understood by the receiving stations and then transmitted. This technique is relatively straight forward, but the requirement for one box per set of stations makes the 'black box' approach expensive.

Another method of implementing a protocol convertor is a software program which accomplishes the same effect as the 'black box.' The signals from the stations are passed through a processor running a protocol conversion program. The program manipulates the signals into protocols understood by the receiving station. This technique is somewhat more flexible than the 'black box,' because of the accessibility of the conversion program stored in software. But software programs also require extensive memory and tend to slow down the conversion process. Depending on the source of the software, the cost is comparable to the 'black box' method.

A third option for implementing a protocol convertor is the use of network-based protocol conversion services, such

as GTE's Telenet and McDonnel-Douglas's Tymnet. These services receive signals from various stations, and convert them to a network standard protocol. The signals are then routed to the network processor nearest the receiving station. At the nearest network processor, the signals are converted from the network standard protocol into a protocol understood by the receiving station and transmitted. Network-based services are only appropriate for systems using many widely dispersed facilities which have access to phone lines.

These are just a few examples of the different technologies and techniques available to accomplish protocol conversion. There are a multitude of combinations of these methods which are also used, such as the combination of software and hardware techniques into a 'firmware' approach.

C. LACK OF STANDARDS

There are as many different standards for protocols as there are methods to implement them. Fortunately, most of these standards are similar in format, timing and methods of conveying information. These similarities are due in part to the basic structure required of a digital communication protocol and in some cases, one common source for many different standards. Despite minor differences, most modern protocols are designed around basically the same frame format. The variations between similar protocols have originated where protocol specifications have been interpreted differently.

Many of the popular protocols are adaptations by the standardization organizations of the same basic protocol. For example, SDLC (Synchronous Data Link Control) which was first developed by IBM (International Business Machines) was modified by the American National Standards Institute (ANSI) to ADCCP (Advanced Data Communication Control

Procedure). The International Standards Organization (ISO) also modified SDLC to become HDLC (High-level Data Link Control). The Comité Consultatif Internationale de Télégraphique et Téléphonique (CCITT) then modified HDLC to become its LAP (Link Access Procedure). Subsequently, the CCITT modified LAP to become LAPB, and integrated it into the X.25 network interface standard. [Ref. 2:p. 168]

Tanenbaum sums up the state of standardization within the digital communication community in his text on computer networks:

> "The nice thing about standards is that you have so many to choose from: furthermore, if you do not like any of them, you can just wait for next year's model." [Ref. 2:p. 168]

With this multitude of different standards and consequently different protocols, the best immediate solution is a parameterized protocol convertor. A parameterized protocol convertor is flexible enough to make allowances for the small but significant differences between the popular protocols, yet fast enough to avoid becoming a bottleneck in the system. Until protocol standardization is established nationally and then internationally, the need for a fast, flexible protocol convertor will exist.

D. INTEROPERABILITY

Variations in protocols and other incompatibilities between communication systems are commonly referred to as interoperability problems. The effects of these interoperability problems can be observed in three separate areas:

- The military services.
- The home computer market.
- The business world.

Interoperability is defined in military terms as:

> "The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together."[Ref. 6]

This 'ability to provide and accept services' is tantamount to compatibility. In effect it means configuring and equipping forces in such a way that they are able to share resources. These resources range from tangible goods such as ammunition, spare parts and POL (Petroleum, Oil and Lubricant) products to less substantial items such as intelligence information, messages and fire support coordination measures. With the introduction of digital communication systems and the extensive use of computers to handle information in the military services, the challenge of interoperability has spread to the computer communication field.

Protocol incompatibility is a major source of problems between the military services. With the number of technologically advanced communication systems being developed and acquired by the different services, the maintenance of a standard communication protocol between them is nearly impossible. Some systems have been deemed adequate for missions they were not originally designed for. The subsequent revelation of incompatibilities with other systems involved with the same mission is usually too late for engineering development changes.

Most of the interoperability problems caused by the use of various protocols can be solved by the implementation of a parameterized protocol convertor. Systems that were not designed to share information could still communicate despite the use of different protocols. Weapon systems capability would not have to be sacrificed for compatibility with other systems or other Services.

Another source of interoperability problems within the military services is the Department of Defense systems

acquisition policy. The military services are directed by Congress to purchase their systems from the civilian industrial base; in direct contrast to the government owned arsenals of previous years. The various contractors and subcontractors employed to build these systems are at different levels of technological maturity, and consequently design systems using different communication protocols.

The need for a flexible protocol convertor is more wide spread than just the military environment. As the home computer market has expanded, the number of uses of home computers is growing also. Home computer owners can now communicate with banks for their account status, access data bases for information on a multitude of subjects, and use other "on-line" services, such as electronic mail. No one standard protocol has been established for the home computer communication market. There are several that enjoy varied levels of popularity, such as XMODEM[1], KERMIT[2], and MNP[3] but they are not compatible with each other [Ref. 7].

It is too late to set a single communication protocol standard through out the home computer market. A significantly large number of home computers and MODEMS[4] have already been purchased supporting various communication protocols; consolidating them would be impossible. The next best solution is a protocol convertor that would make the individual choice of communication protocol insignificant.

---

[1]A widely used error-checking protocol for microcomputers which has been placed in the public domain.

[2]A protocol developed by Columbia University for communications among microcomuputer, minicomputers and mainframes.

[3]Microcom Networking Protocol, a file transfer protocol developed by Microcom Inc.

[4]MODulator-DEModulator: a device which modulates and demodulates digital signals onto and off of phone lines.

Another demand for a flexible protocol convertor is in the business communication network arena. Some corporations are finding it more economical to purchase network system components from different manufacturers rather than tie themselves to one product line, or one vendor for their computing needs. The main reasons for using products from multiple suppliers are reduced cost, flexible hardware and software upgrades, and access to advanced technology [Ref. 8:p. 148]. Other corporations in a hasty effort to obtain computing capability, have amassed a varied assortment of computers all supporting different communication capabilities and protocols. Whatever their source, these system differences come to bare when an attempt is made to tie the various systems into one network. The lack of a common protocol among the numerous system manufacturers is a major obstacle to be overcome in the networking arena.

E. SUMMARY

The mutual set of rules and conventions which communication systems share in order to 'speak the same language' is called a protocol. These protocols solve operating problems in the areas of: framing, error control, sequence control, transparency, line control, time-out control, and start-up control. There is a multitude of different protocols, each with slight, but significant variations. When two systems do not share the same protocol, a special filter called a protocol convertor is required to enable them to communicate. The protocol convertor establishes a data path between communication systems despite differences in speed and message formats, and errors introduced by the communication medium. There are numerous techniques and technologies for implementing protocol convertors, each with its own advantages and disadvantages.

A parameterized protocol convertor is flexible enough to make allowances for the small but significant differences between the popular protocols, yet fast enough to avoid becoming a bottleneck in the system. There are extensive uses for a parameterized protocol convertor. Computer communications users from all walks of life would benefit from the removal of the restrictions imposed by incompatible protocols. The military would see an end to many of its interoperability problems. The home computer user could access an sizable number of different on-line services. The business computer user would be freed from the limited selection of a single equipment supplier. Until protocol standardization is established nationally and then internationally, the need for a fast, flexible protocol convertor will exist.

III. CONFLICTING REQUIREMENTS OF SPEED AND FLEXIBILITY

Protocol conversion for modern communication systems generally necessitates a compromise of speed and flexibility requirements. While protocol conversion must be accomplished at sufficient speeds to avoid becoming a bottleneck in the system, the conversion process must also be flexible enough to accommodate variations between implementations of similar protocols. Unfortunately, the concurrent implementation of these two conflicting performance specifications is not directly obvious. The engineering trade-offs between speed and flexibility call for a careful analysis of the desired speed capabilities and the required flexibility specifications.

Use of the traditional control flow architecture can prove to be a detriment to the effective implementation of a fast but flexible protocol convertor. Microprocessor controlled logic and the incurred dynamic flexibility reduce the speed of the conversion process. Alternate methods, similar in a limited sense to a data flow architecture, offer promising possibilities of an increased speed of operation while maintaining an adequate degree of flexibility.

A. ENGINEERING TRADE-OFFS BETWEEN SPEED AND FLEXIBILITY

One of the most interesting aspects of the engineering analysis of a problem is the comparative weighting of different features or capabilities. Various applications require an emphasis on different attributes of a design, many of which are in contention with each other. In a protocol convertor, the major conflicting attributes are a high speed of operation and an extensive degree of flexibility. A high speed of operation can be defined as

sufficient speed to avoid becoming a bottleneck in the system. Flexibility can be defined as adaptability to changes or variations in the protocols. It can be divided into two broad categories, dynamic flexibility and static flexibility.

For the purposes of this paper, dynamic flexibility is defined as the capability of a device to alter the variable parameters of its function, while the operation is in progress. This extensive flexibility is inherently dependent on a control flow architecture, and the implementation of algorithms in software vice hardware. A control flow architecture supports the comparison decisions, branching and jumping capabilities of the controlling instructions or program. These capabilities permit a device to control its own instruction sequence and to alter the flow of an operation already in progress, to a limited degree.

In contrast, static flexibility is more limited and is defined as the capability of a device to alter the variable parameters of its function, but not while the operation is in progress. Static flexibility does not require the generality of a control flow architecture, and an architecture more compatible with the specific requirements of the application can be exploited. A device designed with static flexibility implements its algorithms in hardware vice software, and is flexible only in that parameters of the operation can be set before use. If changes are necessary, the process must be halted, the changes made, and the process restarted.

The implementation of a device with dynamic flexibility sacrifices some of the otherwise possible speed capabilities of the device. The use of software to implement algorithms reduces the overall speed of operation of the device, because of the requirements to interpret the instructions stored in software and fetch the operands.

Implementation of the same device with the more limited static flexibility will typically indicate a marked increase in speed of operation. To use static flexibility, the algorithms of the application must be somewhat limited in scope and implemented entirely in hardware. Which degree of flexibility and corresponding speed of operations is used is dependent on the requirements of the application of the device. For example, protocol conversion requires only limited flexibility, but it does require sufficient speed to avoid being the bottleneck in the communication link.

The implementation of the most time consuming aspects of a process in hardware is termed functional specialization. While functional specialization may provide an increase in speed of operation, it also requires a trade-off in the form of a restricted application of the system. A system implemented with functional specialization is limited in its flexibility of application to one specific area of operations [Ref. 9:p. 201]. The concept of a parameterized protocol convertor is that of a dedicated machine in that it is designed to perform protocol conversion only. Sufficient flexibility for general application is forfeited for an increased speed of operation.

Bracker [Ref. 10], in his article on the current protocol vendor offerings, lists forty-three different devices which convert from protocol A to protocol B. Twenty-five of these devices are hardware and or software systems designed to convert between two specific protocols. They do not provide for any combinations of protocols other than those specified by the manufacturer, and only limited variations of the two protocols supported.

The next major group of protocol convertors listed are front-end processors; devices which are sold as protocols convertors but also have some user programmable capability.

Even with the flexibility provided by their programming capability, they are still listed as only being able to convert between two different protocols. The programming capability of these devices does permit changes in the system to account for variations of the two protocols supported. The rest of the protocol convertors listed provide specific services such as conversion from full to half-duplex and emulation of specific data terminals. Of the forty-three listed as capable of converting protocol A to protocol B, only three are strictly hardware systems.

The currently available protocol convertors offer dynamic flexibility only. They use a software implementation of the conversion algorithms to achieve the desired levels of flexibility with a consequential reduction in the speed of conversion. Protocol conversion is one process where static flexibility should be sufficient. The parameters of a protocol are not changed while the convertor is in operation, only when the system protocol is altered in some way. Limiting the implementation of a protocol convertor to the static flexibility of hardware implemented algorithms should increasing the speed of the conversion operation.

B. REQUIREMENTS FOR SPEED

The requirement for fast protocol conversion is driven by the desire for the conversion process to be invisible to the user. Communicating with a separate system using another protocol and a protocol convertor should not appear any different to a user than communicating with a separate system using the same protocol. A hardware implementation of the protocol convertor is usually required to maintain a sufficient speed of conversion.

An acceptable speed of conversion is tied to many communication link hardware specifications. These specifications include the communication capabilities of

the system with other systems and the communication channel bandwidth limitations. In order for the protocol convertor not to be the bottleneck in the communication system it must operate at least as fast as the slowest piece of hardware involved with the communication link. This minimum requirement is just that, a minimum requirement and should not be taken as a design goal. Both the communication capabilities of modern systems and the bandwidth of communication channel technologies are being improved at a steady rate.

Siewiorek et al. [Ref. 11] uses Kiviat graphs to summarize the major performance parameters of several popular systems. One dimension of the Kiviat graphs is dedicated to the systems communication capabilities with other computers. The source of the system communication speed limitations are typically due to limited system bus capabilities or slow CPU clock speeds relative to communication speeds. Table 3.1 lists several systems and their capabilities for communication with other systems.

TABLE 3.1 COMPARATIVE SYSTEMS COMMUNICATION CAPABILITIES

SYSTEM	BITS/SECOND
VAX 11/780	512×10^3
IBM 370 Model 155	784×10^3
BSP (Front End B7800)	12×10^6
CYPER 170	16×10^6
CYPER 205	50×10^6
CRAY 1	50×10^6

Another aspect of the communication link hardware specifications which effects the desired speed of protocol conversion is the communication channel bandwidth limitations. Figure 3.1 and 3.2 illustrate the data rate

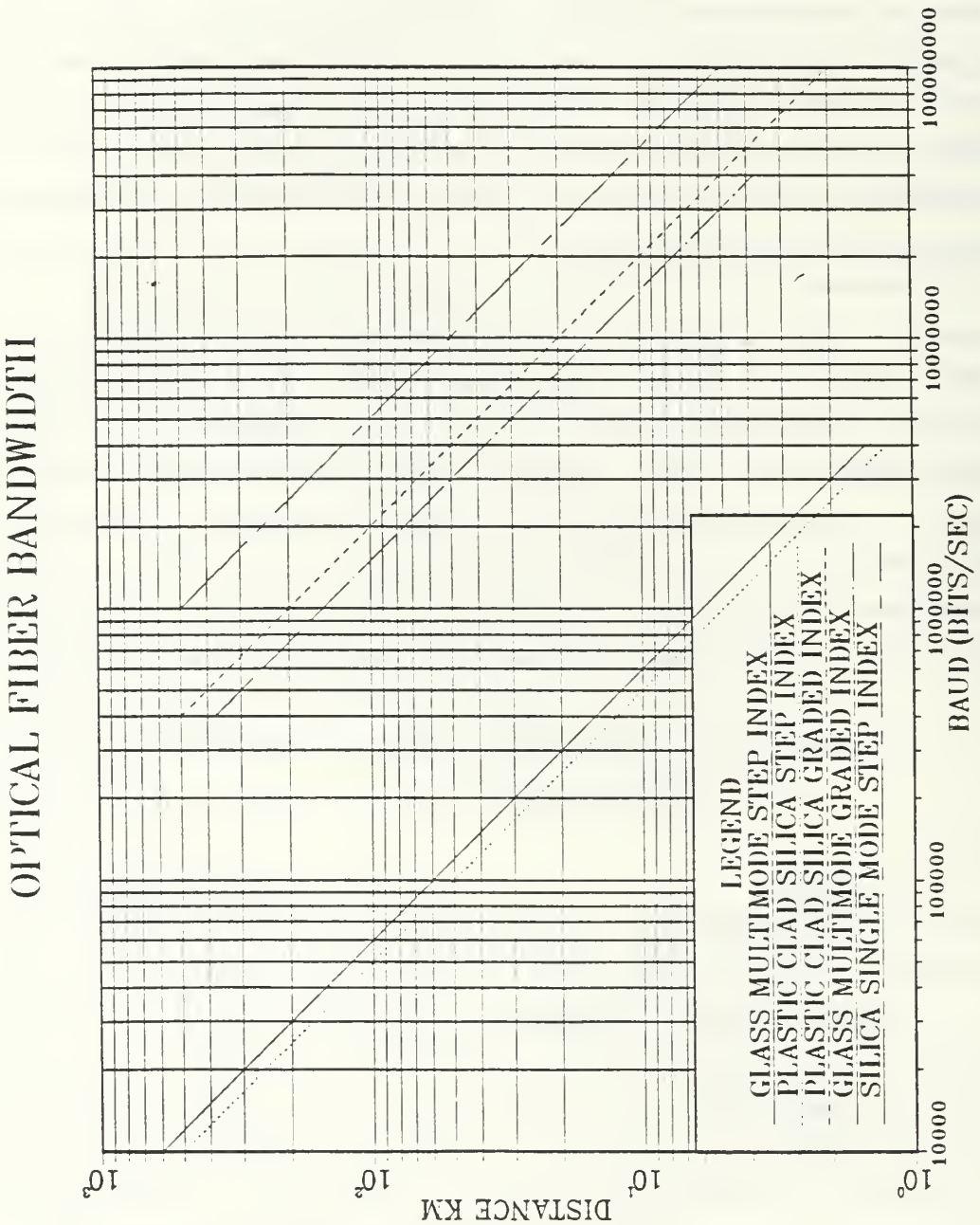

Figure 3.1 Optical Fiber Bandwidth

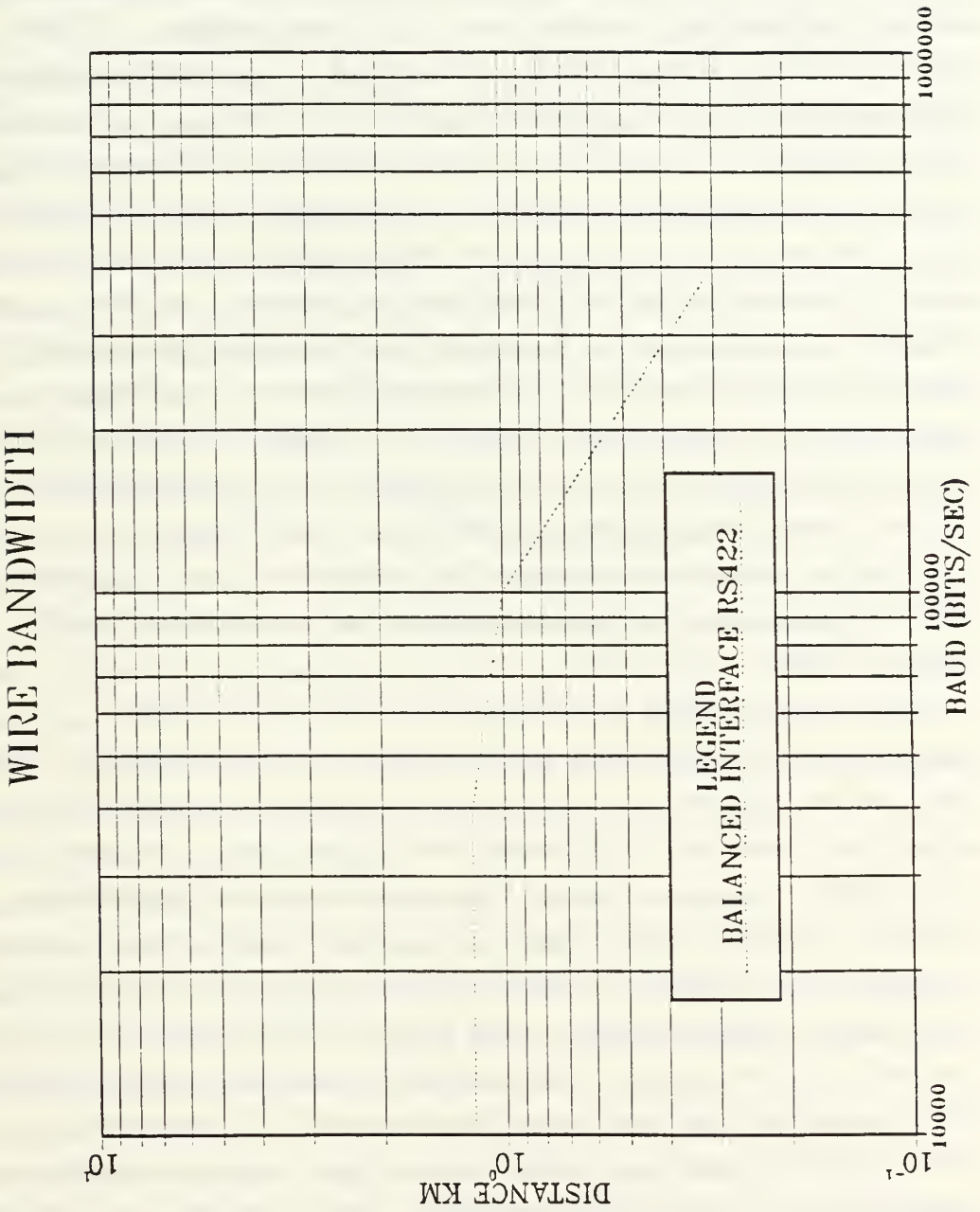

Figure 3.2 Wire Bandwidth

versus distance relationships for several variations of two of the common digital communication channel technologies; fiber optics and copper wire.

The latest technological breakthrough in communication channel technology is the use of fiber optics. Fiber optics offer a far greater potential transmission bandwidth than metallic cable systems or radio systems. A coaxial cable system is limited to approximately 500 megahertz of transmission bandwidth, and a millimeter wave wideband radio system to approximately 700 megahertz. Currently the bandwidth available to fiber optic systems is the range of several gigahertz over a few kilometers and hundreds of megahertz over tens of kilometers without intervening repeaters. In the near future the usable fiber optics system bandwidth will be extended further towards the optical carrier frequency (10^{13} Hz - 10^{16} Hz) to provide an information carrying capacity far in excess of that obtained using copper cables or a wideband radio system. [Ref. 12:p. 7]

One significant limitation to the available communication bandwidth of a fiber optics system is the electronics which are required to support communications using the channel. Although the fiber optic channel itself can handle transmissions of several hundred megahertz, the circuitry used to modulate the signals onto the channel is limited to the speed of the current circuit technologies. The common, commercially available TTL[1] circuitry is limited to the area of 20 megahertz and the more expensive ECL[2] operates typically in the area of 70 megahertz. Until circuit design technologies can achieve the same bandwidth as fiber optic systems, the sizeable bandwidth available can be utilized only with multiplexing systems, which combine many different signals onto one channel.

---

[1]Transistor-Transistor Logic

[2]Emitter-Coupled Logic

The speed considerations for protocol conversion in this case are for one single data stream, without multiplexing.

Current communication channel bandwidth limitations are no reason to reduce efforts to provide the fastest protocol conversion services possible. As technologies mature, faster communication channels will be developed, until the protocol convertor will eventually become the bottleneck in the communication link.

If sufficient communication channel bandwidth and processor communication capabilities are available, processing of the protocol becomes the major detriment to fast protocol conversion. A protocol convertor that can not keep pace with the data being input into it requires some form of data storage. If the size of the storage buffer is not sufficient to absorb the difference between data reception speed and data conversion speed, buffer overflows occur and increase the processing delay even more. Any buffer overflow requires the re-tranmission of the data, again slowing the system's effective throughput. To overcome these obstacles, efforts are made to streamline the data manipulation by simplifying the conversion process and using the parallel constructs of a data flow architecture. [Ref. 13:p. 2]

Another aspect of the required speed capabilities is the conversion between the serial data used for communication between systems and the parallel data used internal to the system. Since most digital communication traffic traveling any major distance is passed over a single channel per transmitter medium, the data must be transmitted serially. This is in contrast to the short distance communication between a system and its peripherals usually accomplished on parallel lines. The serial transmission of data requires very high bit rates to keep up with the fast parallel movement of data within modern systems, and to maintain a reasonable throughput.

Joshi and Iyer [Ref. 13] use the illustration of a funnel as a tool to help describe the conversion between parallel and serial data and the resulting increase in speed of the data. As the data is received from a system in parallel form into the wide mouth of the funnel, it is converted into serial form and passed to the transmitting medium at the constricted funnel end. An analogy is made between the data and a fluid in the funnel, where the data moves much more rapidly at the constricted end of the funnel than at the wide mouth of the funnel.

Any processing performed on a data stream can be viewed as a perturbation which causes turbulence in the funnel, because it slows down the movement of data. Where in the funnel the data is processed determines the extent of the effect of the turbulence. If the processing occurs in the wide (parallel) section of the funnel, the effect of the turbulence is minimal. This is in contrast to processing in the narrow (serial) section where any degradation in speed is of major concern. Processing in the high speed narrow section of the funnel requires high speed and consequently expensive hardware. In order for less expensive techniques to be used effectively, the configuration of the processing and the conversion architecture must be carefully defined.

C. REQUIREMENTS FOR FLEXIBILITY

The requirement for a flexible protocol convertor is driven by several factors related to current protocols and their standardization. These include:

- The many variations of currently popular protocols.
- Inevitable changes in current protocols.
- The development of new protocols as the standards are approved.
- A significant reduction in hardware redesign requirements.

The desired degree of flexibility in a parameterized protocol convertor requires the use of a changeable parameter store, such as software or firmware. This is in contrast to a protocol convertor implemented totally in hardware which can not be readily adjusted to any alterations in the conversion process.

A limited degree of flexibility is required to account for the subtle differences in various implementations of the currently popular protocols. These differences are the result of varied interpretations of the protocol standards and the implementation of these differences by various system manufacturers. An example is the development of LAP from SDLC described previously.[3]

As communication techniques and technologies advance, there will be inevitable changes in the protocols currently installed on major communication systems. These changes will require a modification of the parameters of a protocol convertor. If the changes are anticipated, and sufficient flexibility built in, the protocol convertor will not become obsolete any faster than the technology of the system it supports.

Once the adaptability of the current protocols has been exhausted, new ones will have to be developed. These new protocols will include the latest state-of-the-art techniques of communication protocol technologies. The chances of anticipating sufficient flexibility requirements to absorb the changes in this situation is doubtful, but nevertheless a worthy goal.

As new protocols are accepted as standards, a flexible parameterized protocol convertor will be adaptable to the changes without major hardware reconfigurations, or modifications. Under the concept of static flexibility, the parameters most likely to be altered or adapted in a new protocol are maintained in a changeable control store.

---

[3]See p.15 - protocol standards.

Only the apparently consistent operation algorithms are implemented in hardware. Unless the development of a new protocol significantly alters the basic operation of protocol conversion, the only changes should be in the easily accessible parameters.

D. WHY NOT CONTROL FLOW ARCHITECTURE

A major consideration in the design of a system is the relation between the algorithms to be implemented and the architecture to be used. When there is a correlation between the algorithm and the architecture, a synergistic effect can be expected. Both the algorithm and the architecture seem to perform better by their relationship with the other. The algorithms involved in protocol conversion are not suited to the control flow architecture they are currently implemented with.

The traditional control flow architecture or Von Neumann architecture is know for its generality and flexibility in that it supports a large variety of programming languages and styles with reasonable effectiveness. Its flexibility stems from the control flow structure which allows the programmer, the compiler, and or the interpreter direct control over the low level machine operations when necessary [Ref. 14:p. 594].

Control flow architectures are also know for their implicit sequential nature. There is a single thread of control passed from instruction to instruction, resulting in explicit transfers of control from one instruction to the next. The instructions have limited control over their own sequence of execution. [Ref. 15:p. 734]

Hwang and Briggs [Ref. 15] list several identifying characteristics of a control flow architecture:

- Data is passed between instructions via references to shared memory cells.
- The flow of control is implicitly sequential.

- Program counters are used to sequence the execution of instructions in a centralized control environment.

These characteristics provide a high degree of flexibility, but not without some significant trade-offs. There is a substantial cost in speed of operations in order to allow an almost universal applicability of the architecture.

The majority of the available protocol convertors use both hardware and software in a microprocessor controlled logic system. These logic systems resemble the traditional control flow architecture with the program or instruction sequence stored in software. The conversion speed of these units is directly related to the controlling microprocessor speed. Both the system clock frequency and the rate at which the microprocessor can sequence through its instruction cycle limit the speed at which conversion can be realized. Although fast enough for most applications, the sequential nature of their operation, the long instruction execution times, and the centralized program control result in a system which is too slow to meet the critical time constraints of protocol conversion [Ref. 16:p. 13].

A study of comparative architectures would indicate that the more aspects of a system are implemented in hardware, the faster the speed of the system. This increase in speed is offset by a corresponding reduction in flexibility. One example is the Intel 8087 Numeric Data Processor (NDP) used in conjunction with the Intel 8086 or 8088 microprocessor. The 8087 NDP performs only one basic type of function; arithmetic and transcendental operations on integers and real numbers. There are only fourty-eight instructions available in the 8087 instruction set, and they are all oriented towards numeric operations. The 8087 NDP performs these arithmetic and transcendental operations

at a five to ten fold speed increase over the more flexible processors it supports [Ref. 17:p. 40].

Flexibility realized from the storage of instructions in software is one example of dynamic flexibility. The instruction flow of a system under software control can be altered as the system is operating. For example, most languages implemented on control flow architecture machines support the use of comparison decisions, and subsequent branching. These branches effectively change the flow of instructions dependent on either the results of internal calculations or external inputs. This dynamic flexibility, afforded by the storage of instructions in software, is a desirable asset in many situations requiring varied applications of the same design.

Although desirable in some situations, a fixation with flexibility can prove fatal to speed capabilities. In general terms, the more flexible a system, the slower the system is in operation. The possibility of several applications is typically gained at the expense of the speed of operation of the variations.

E. DATA FLOW ARCHITECTURE

A more promising approach to implementing a fast yet flexible protocol convertor is the use of limited aspects of a data flow or data driven architecture Where a control flow architecture is oriented towards the sequential interpretation and execution of instructions, a data flow architecture exploits parallelism by executing instructions as the required operands for the instruction become available, regardless of the order of the instructions. The number of different operations being concurrently executed in a data flow architecture is limited only by the hardware resource availability [Ref. 15:p. 29].

According to Gajski et al., [Ref. 18], a data flow model of computation is based on two principles:

- Asynchrony - operations are executed when and only when the required operands are available.
- Functionality - all operations are functions without side effects, that is, any two operations can be executed in either order or concurrently.

In a data flow architecture, many instructions can be executed simultaneously and asynchonously. The scheduling and synchronization of concurrent activities are built in at the hardware level, enabling each instruction to be treated as an independent concurrent action [Ref. 14:p. 487].

Generally, to increase the speed of operation of a control flow architecture system, there must be an increase in the speed of the individual components of the system. Just adding more components is usually of limited value. In contrast, a data flow architecture system benefits from an increase in the number of processors, up to a limit where the communication delay between the processors is greater than the processing time of a single processor. In a parameterized protocol convertor, increasing the number of processors correlates to increasing the number of simultaneous operations performed.

The parameterized protocol convertor is not a true implementation of the data flow architecture. The data flow architecture is usually implemented as a general purpose, programmable system. The parameterized protocol convertor is a dedicated machine, designed for one application only. Both the parameterized protocol convertor and a data flow architecture:

- Exploit parallelism.
- Instructions (or operations) are executed when the operands become available.
- Instructions (or operations) are not ordered.

The parameterized protocol convertor uses operations stored in hardware instead of the instructions of a true

data flow architecture. These operations are essentially algorithms implemented directly with programmable logic arrays (PLA) and random logic.

The data path implementation of a protocol convertor is a 'dedicated machine,' in that it only performs the function of protocol conversion. The 'general purpose machine' flexibility of the control flow architecture has been abandoned for the sake of increased speed of operations. Not all flexibility has been sacrificed for this increase in speed, only the dynamic flexibility afforded by the branching abilities of a control flow architecture.

Partitioning a process and devoting a separate processor to each part of the overall process is called functional decomposition. Though not as effective as functional specialization, functional decomposition also produces an increase in the speed of operations. For example, the use of parallel data flow within the protocol convertor assists in obtaining the desired high speed of conversion.

The concept of functional decomposition is applied to the data stream to be processed. As the data stream is received it is copied into parallel shift registers The data is then manipulated concurrently from each shift register. In particular, the determination of transparent data, the error control process, and the determination of the frame limits all take place at the same time.

The determination of how to break the incoming data stream into sections has a major impact of the overall speed of the conversion process. Any increase in speed of operation from an implementation of functional decomposition is dependent on the even distribution of work between the operations, to prevent one of them from becoming a bottleneck in the system.

F. SUMMARY

To be effective, a parameterized protocol convertor necessitates a balance between conflicting speed and flexibility requirements. Adequate flexibility must be provided to account for variations in the protocols supported, and sufficient speed of operations is required to avoid becoming a bottleneck in the system. The currently available protocol convertors offer a limited degree of flexibility, but their implementation with software systems and control flow architectures reduce the possible speed of conversion.

The requirement for fast protocol conversion is in comparison to the communication capabilities of the system. Systems are typically limited by restricted bus capabilities, CPU clock speeds and communication channel bandwidth limitations. The protocol conversion process should not be a bottleneck in the system architecture.

The requirement for a flexible protocol convertor is driven by the lack of standardization of protocols. There are many variations of the popular protocols, and even these change periodically. Some flexibility must be allowed if the protocol convertor is to remain unaffected by the changes in the protocols supported.

The extensive flexibility of control flow architectures can be a detriment to a specific application with major speed requirements and limited flexibility requirements. Alternative architecture concepts, such as the data flow architecture can be employed in a protocol convertor to assist in the achievement of sufficient speeds of operation.

IV. PROPOSED ARCHITECTURE FOR PROTOCOL CONVERSION

A. AN EXAMPLE

A typical example of protocol conversion will help illustrate the basic concepts of the parameterized protocol convertor. Two stations of a communication system are required to exchange data at a high rate on a half-duplex communication channel in a serial synchronous mode. Station A uses DDCMP (Digital Data Communications Message Protocol), a typical byte count protocol, and Station B uses SDLC (Synchronous Data Link Control), a typical bit oriented protocol. Neither station has an internal protocol conversion capability; both of them must rely on external convertors to exchange information with stations supporting other protocols.

The data to be exchanged between Station A and Station B consists of relatively short frames on the average of forty to fifty bytes each. The Station A DDCMP frame format is illustrated in Figure 4.1.

SYN	SYN	CLS	CNT 14	FLG 2	RSP 8	SEQ 8	ADD 8	CRC1 16	INFO	CRC2 16

Figure 4.1 DDCMP Frame Format

Where SYN is the synchronization character, CLS is the class of the frame, CNT is the byte count, FLG is a quick synchronization or select flag, RSP is the response to the last frame, SEQ is the sequence number of this frame, ADD is the address, CRC1 is the header block check characters, INFO is the information field and CRC2 is the information field block check characters [Ref. 3:p. 158]. The numbers

under the acronyms are the length of the fields in bits. The information field can be up to 16,363 bytes in length. The first nine fields of the frame contain information relative to the system protocol. Only the information field contains the data required by the user at Station B.

Figure 4.2 illustrates the SDLC frame format.

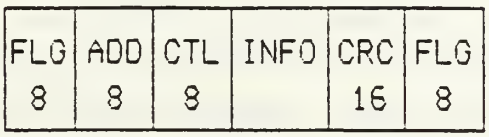

Figure 4.2 SDLC Frame Format

Where FLG is the synchronization flag, ADD is the frame address, CTL is a control byte, INFO is the information field, CRC is the frame block check characters and FLG is the synchronization flag again [Ref. 3:p. 164]. The numbers under the acronyms are the length of the fields in bits. The information field can be any number of bits. The first three fields of the frame contain information relative to the system protocol. Similar to the DDCMP frame format, only the information field contains the data required by the user at Station A.

Two parameterized protocol convertors are required to support communication between Station A and Station B. The parameterized protocol convertor installed with Station A is set for a byte count input protocol and a bit oriented output protocol. The parameterized protocol convertor installed with Station B is set for a bit oriented input protocol and a byte count output protocol.

To aid the protocol convertor in determination of the frame boundaries, detection of any transparent data, and in error control, several aspects of the two protocols must be specified. These include, the code type used, the bit sequence used as a synchronization character, the length of

the header and which CRC generator is used. The parameter inputs for both convertors are set as indicated in Table 4.1 for the byte count protocol parameters and Table 4.2 for the bit oriented protocol parameters.

TABLE 4.1 BYTE COUNT PROTOCOL PARAMETER SETTINGS

Code Type - ASCII
Synchronization Character - 00010110
Header Length (after synchronization) - 8 bytes
Byte Count Length - 14 bits
First Bit of Byte Count - bit 9
CRC Generator - CRC-16

TABLE 4.2 BIT ORIENTED PROTOCOL PARAMETER SETTINGS

Code Type - ASCII
Synchronization Flag - 01111110
Header Length (after synchronization) - 2 bytes
CRC Generator - CRC-CCITT

Control codes are another aspect of the two protocols which must be specified. The two stations exchange information about their status and the condition of received frames through the use of control codes. The control codes must be translated so that each station only receives control codes it will recognize. The required control code translations for both protocols are indicated in Table 4.3.

The exchange of data proceeds as follows. Station A sends an initialization or enquiry message to Station B in the form of a DDCMP Start Message Control Code. The frame is received into the Station A parameterized protocol convertor where the DDCMP Start Message Control Code is

transposed into the SDLC Set Initialization Mode Control Code and routed to Station B. Station B receives the Set Initialization Mode Control Code and initiates the system specified procedures for frame reception.

When Station B is ready to receive data from Station A, Station B responds to the initialization message with the SDLC Nonsequenced Acknowledgment Control Code. The frame is received into the Station B parameterized protocol convertor and transposed into the corresponding DDCMP control code, Start Acknowledge. Station A receives the Start Acknowledge Control Code and interprets it as an indication that Station B is initialized and ready to receive a data frame.

Once both stations have indicated that they are ready to exchange data, Station A sends its first data frame. The frame is received into the Station A parameterized protocol convertor for conversion to the SDLC format. The frame manipulation includes determination of the length of the information field of the frame, detection of any data which should be made transparent to the SDCL control code detection circuity, and error control. As the frame is manipulated, it is passed out of the Station A parameterized protocol convertor to Station B. At no time is more than one byte of the frame stored in the Station A protocol convertor.

Once the entire frame is received at Station B, it is checked for errors. For purposes of illustration, the block check characters indicate an error in the reception of the frame. The SDLC Reject command code is the sent back to Station A by Station B indicating a request for re-transmission of the last frame. The SDLC Reject command code is transposed to the DDCMP Negative Acknowledge command code within the Station B protocol convertor and passed to Station A.

TABLE 4.3 CONTROL CODE TRANSLATION TABLE
X INDICATES DON'T CARE.

Byte Count Protocol Control Codes	Bit Oriented Protocol Control Codes
Start Message 00000101 00000110	Set Initialization Mode 1101X000
Start Acknowledge 00000101 00000111	Nonsequenced Acknowledgment 11001110
Negative Acknowledge 00000101 00000010	Reject 1001XXXX
Positive Acknowledge 00000101 00000001	Receive Ready 1000XXXX

Upon receipt of the Negative acknowledge, Station A re-transmits the initial data frame. This time the frame is received without errors and Station B replies with the SDLC Receive Ready command code. The SDLC Receive Ready command code is transposed by the Station B convertor into the DDCMP Positive Acknowledge command code and sent back to Station A. Upon receipt of the Positive Acknowledge command code, Station A sends the next frame. The processes cycles through the data frame transmission and acknowledgment sequence until all the frames are received without errors by Station B.

This is an example of conversion between one possible combination of input and output protocols available with the parameterized protocol convertor. Any combination of the three framing technique protocols is available. A fast, yet flexible design is required to allow conversion between any combination of input and output protocols while at the same time avoiding becoming a bottleneck in the communication system.

B. SYSTEM BLOCK DIAGRAM DESCRIPTION

A block diagram of a system architecture designed to provide this fast, yet flexible protocol conversion service is included as Figure 4.3. The major components of the system are a data path controller and two protocol conversion units for each protocol supported. Of these two conversion units per protocol, one interprets the incoming data stream, and the other manipulates the outgoing data stream. The central data path controller acts as the coordinator of the conversion process, directing data between the different protocol conversion units and the external systems.

There are three types of inputs into the system: the protocol select inputs, the parameter inputs and the data inputs. The data inputs are the only dynamic inputs into

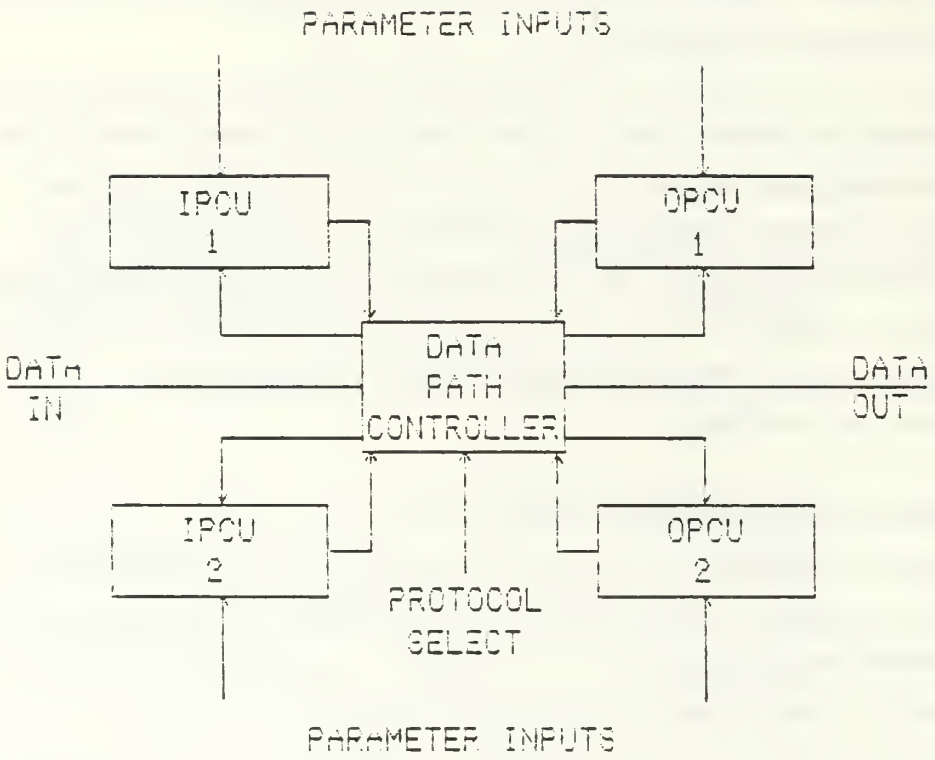

Figure 4.3 System Block Diagram

the system. They consist of the incoming frames to be converted into a different protocol and passed on to another communication station.

The protocol select inputs and the parameter inputs are static inputs. They are interpreted at initial system start-up and set until the system is turned off or reset. The static inputs are comparable to the firmware used by some system manufacturers to store the command sequences and instructions in ROM (Read Only Memory).

In operation, the input data is taken into the data path controller. The previously determined data path directs the data sequence to the selected input protocol conversion unit. The data sequence is manipulated and passed back to the data path controller for redirection to the output protocol conversion unit. Here the data is manipulated again and passed back to the data path controller. The converted data sequence is then passed out of the parameterized protocol convertor to the receiving communication station.

The protocol select inputs determine which protocol the convertor should use to interpret the input data sequence, and which protocol the convertor should use to produce the output data sequence. These inputs control the course of the data path. The use of a controlled data path permits substantial flexibility in the selection of the input and output protocols.

C. PROTOCOL CONVERSION WITH HARDWARE

The protocol conversion process is centered both conceptually and physically around an internal virtual protocol. The input protocol conversion unit interprets the input data according to its parameter inputs and converts the pertinent aspects of the input data into the virtual protocol format. The virtual protocol formatted data is then transferred back to the data path controller

for direction to the selected output protocol conversion unit. The output protocol conversion unit accepts its input in the virtual protocol format, and converts the data to the desired output protocol.

Selection of the virtual protocol is the key to the simplification of the conversion process. This simplification in turn allows the use of a less sophisticated but faster logic system. A complex virtual protocol which is only used internal to the protocol convertor reduces the amount of dynamic flexibility required in the conversion process. If the majority of variations between protocols can be represented with the internal virtual protocol, a minimum of dynamic external inputs to the conversion operation are needed. Any dynamic inputs to the conversion process can be viewed as probable sources of delay because of the requirement to interpret the inputs while the process is being executed.

Conversion between relatively similar protocols is straight forward; only minor differences have to be accounted for. Conversion between protocols using different framing techniques is more complex and subsequently more difficult to implement with a virtual protocol. The different framing technique protocols vary on their frame formats and how information is specified.

To aid the virtual protocol selection process, the protocols and their compared functions can be viewed as a three dimensional array. See Figure 4.4. The array has the functions to be compared on one dimension (Y axis), the three different framing technique categories on another dimension (X axis), and the same message framing technique category protocols on the third dimension (Z axis).

The differences between various implementations of the same framing technique category can be parameterized. These differences include minor variations of the same basic function. For example different implementations of

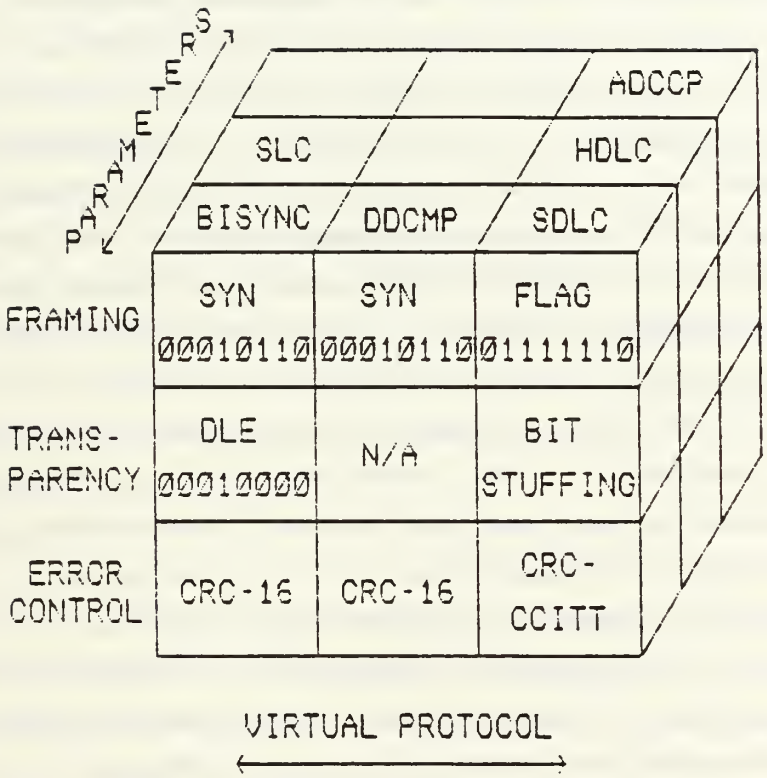

Figure 4.4 Virtual Protocol Selection

character oriented protocols may use different bit sequences for the synchronization character used in framing. Static values for each of these parameters are input into the two protocol conversion units in use.

The common aspects of the various framing technique categories can be implemented in the internal virtual protocol, keeping in mind the popular implementations of each framing technique. For example, both bit oriented and character oriented protocols use a special flag or character to mark the beginning and end of frames. A similar flag is implemented in the internal virtual protocol to accommodate the sub-function of framing.

The process of protocol conversion requires some, but not all of the sub-functions performed by the two protocols involved. All of the sub-functions are performed by the communication systems external to the protocol convertor, and many of the protocol sub-functions require sophisticated circuitry. Redundant implementation within the protocol convertor of any sub-function not implicitly required for the conversion process will reduce the speed of conversion from its optimum.

The protocol convertor performs only those sub-functions of the protocol necessary to convert from one protocol to another. For example, the sequence numbers of the exchanged frames are tracked by the external communication systems. The sequence numbers are an indication to the external communication systems of the order of the exchanged frames. The sequence number of a frame lost to noise corruption or which was misdirected will not be received. Any subsequent sequence number received will be out of order and will indicate to the receiving communication system that a frame was lost.

To implement sequencing as part of the protocol convertor would require a more sophisticated logic system and subsequently a reduced throughput of data. In keeping

with the filter concept, the protocol convertor manipulates the sequence numbers in the same way as any other data, and passes them though to the receiving station of external communication system. With or without the protocol convertor, it is still the responsibility of the external communication system to interpret the frame sequence number. This includes detecting any out of sequence frames and requesting a re-transmission of the lost frames.

The sub-functions required in the protocol conversion process include:

- Framing.
- Transparency.
- Error Control.

All three of the sub-functions are part of the data link layer of the ISO OSI reference model. It is the data link layer that segments the input data into frames and transforms the raw transmission facility into a communication channel which appears free of transmission errors [Ref. 2:p. 17].

The protocol conversion process requires the framing sub-function because both input and output protocol conversion units must be able to differentiate between the three major framing techniques; character oriented, byte count, and bit oriented. The input protocol conversion unit must recognize where the frames of data and the separate characters within the frames of data begin and end. This recognition is necessary to convert the input data into the internal virtual protocol. The same capabilities are required of the output protocol conversion unit. The output protocol conversion unit must be able to recognize the character and frame boundaries of both the internal virtual protocol and the output protocol.

The framing sub-function is realized by the insertion of special bit sequences in the input data by the transmitting station to indicate the beginning of a frame.

When these bit sequences are detected at the receiving station, the two stations can be synchronized. Knowing the starting bit of the first character, the receiving station can then divide the succeeding data stream into word length groups of bits at the correct boundary points. Since synchronization bit sequences typically occur at the beginning of a block of data, frame-to-frame synchronization is established simultaneously with character-to-character synchronization [Ref. 19:p. 179].

The internal virtual protocol of the protocol convertor uses a bit oriented framing technique. The bit stuffing used by the bit oriented framing technique is the most flexible of the three major framing techniques, because it does not require a set character length in bits. Being the most flexible, the bit oriented framing technique is also the most popular, and using the most popular framing technique in the internal virtual protocol reduces the variation between the average expected input or output protocol. There is a high probability that either the selected input protocol or the selected output protocol or possibly both will be of the bit oriented framing technique. Reducing the difference between the expected input and output framing technique and the internal framing technique decreases the number of variations which must be absorbed within the virtual protocol, and subsequently passed on to the output protocol.

The various protocols of the three different framing techniques delineate frame by defining a special synchronization character. The synchronization character is called a sync in character oriented protocols and byte count protocols. In bit oriented protocols the synchronization character is called a flag.

The major difference between the different framing techniques is selection of the sync/flag character and the method used to determine the length of the data frame.

Character oriented and byte count protocols typically use the ASCII[1] character SYN for synchronization. Bit oriented protocols typically use the bit stream 01111110 as a synchronization flag. Character oriented and bit oriented protocols mark the beginning and end of a frame with control characters or flags. Byte count protocols indicate how many characters are in a frame with a byte count in the frame header.

The sync/flag character is chosen so that its bit arrangement is significantly different from any other anticipated character which is regularly transmitted. The sync/flag character must have an irregular pattern so that any likely combination of characters before or after the sync/flag will not appear to the system as a sync/flag character. For example, the ASCII character SYN typically used by character oriented protocols and byte count protocols consists of the bits: 10010110. An irregular pattern reduces the probability of the communication system synchronizing its operation with the wrong bit sequence. Interpretation of the wrong bit sequence as the synchronization character would lead to the mis-identification of which groups of bits constitute characters and which groups of characters composed frames.

The second sub-function necessary for protocol conversion is transparency. Transparency is the sub-function that permits the transmission of data that would otherwise be interpreted as a control character. The bit sequence of a control character may need to be transmitted within a frame, as binary data, without its usual framing significance. Transparency allows these characters or bit strings to pass through the protocol convertor without triggering the protocol framing mechanism.

---

[1]American Standard Code for Information Interchange is a seven bit plus parity code established by the American National Standards Institute.

Character oriented protocols and bit oriented protocols use a procedure called 'stuffing' to delineate those characters which could be incorrectly interpreted as control characters. In character oriented protocols, a control character DLE is inserted before any byte of data which has the same bit pattern as a control character, but should not be interpreted as a control character. Inserting a DLE into the data stream is called 'character stuffing.' Bit oriented protocols use a similar method to delineate transparent data. A single zero is stuffed into the data stream whenever five successive ones are detected in the data stream. The stuffed zeros prevent the receiver from interpreting binary data within the text field of the frame as the end of frame flag.

In order to function properly, the protocol convertor must be able to recognize transparent data as such. The circuits necessary to strip out stuffed bits and characters must be present in both the protocol convertor and the external communication systems.

The sub-function of transparency is in keeping with the concept of the protocol convertor as a filter with limited flexibility. The control characters are filtered out from the transparent data, interpreted and passed out of the convertor in the output protocol. The character or bit stream to be stripped out by the input protocol conversion unit is one of the parameter inputs, as well as the character or bit stream to be stuffed back into the data stream by the output protocol conversion unit.

The sub-function of error control must be implemented by both the protocol convertor and the external communication system. Error detection is performed by the protocol convertor in order to generate the required block check character for the output protocol. The output protocol conversion units must be able to generate various block check characters as required by the output protocol

parameters. For example, if the input protocol specifies a CRC-16 method of error detection and the output protocol specifies a CRC-CCITT, the protocol convertor must be capable of generating the required block check character from the input data stream.

The detection of any errors in the input data stream must first be accomplished before the expected output protocol block check character is generated. Generating an output protocol block check character without first testing the input data stream for errors would indicate to the receiving station that all frames were error free, at least up until they passed through the protocol convertor. Without error detection on the part of the protocol convertor, corruption of the message data that took place before the input data entered the protocol convertor could not be detected by the external communication system.

The physical location of the protocol convertor relative to the two communication stations will also determine the need for error detection by the protocol convertor. If the protocol convertor is physically located with the transmitting unit of the external communication system, with a minimum length of noise susceptible communication channel between the convertor and the transmitting unit, the number of errors inserted into the data stream before the data stream reaches the protocol convertor will be minimal. The guarantee of a relatively noise free channel for the input data before the data reaches the protocol convertor would remove the necessity for error detection on the part of the protocol convertor. The message data received from the adjacent communication system could be assumed to be error free, and the conversion done without any concern for error detection within the protocol convertor. The only error control capabilities required in this case would be a block check character generation capability by the output protocol

conversion units. The physical location of the protocol convertor anywhere other than adjacent to the transmitting station would require implementation of the full range of the error control sub-function within the protocol convertor, including both error detection and block check character generation.

The sub-functions of line control, time-out control, sequence control and initialization are all employed by the external communication system. They are protocol specific only in the particular characters required to originate the sub-functions. These symbols are converted by the protocol convertor, just like any other message data, with no special significance attached. If the sub-functions are implemented with information only control codes, their conversion is accomplished with a translation table. The inclusion of these sub-functions in the protocol convertor would be redundant.

The external communication system determines the direction of data movement, that is which station is to transmit, and which station is to receive. There is no requirement for the protocol convertor to be engaged in the line control sub-function. A single protocol convertor can only manipulate the data stream in one direction, therefore, two separate protocol convertors would be required for a two-way exchange of data. The use of two independent protocol convertors also allows full duplex[2] operation if the external communication system is also capable of full duplex operation.

The concept of the protocol convertor as a static filter implies that dynamic initialization of the protocol convertor by an input data stream should not be required. The protocol convertor is always ready to operate, with its specifics of operation indicated by parameter inputs.

---

[2]Full duplex is defined as simultaneous two-way independent transmission in both directions. [Ref. 3:p. 306]

The initialization of the stations of the external communication system takes place through the protocol convertor, without the protocol convertor itself requiring any initialization.

D. PROTOCOL CONVERSION UNITS

The input and output protocol conversion units are similar in design and operation. See Figure 4.5. Both are designed with a series of registers, programmed logic arrays (PLA) and control gates. Together, these provide the capability to convert between the input protocol, the internal virtual protocol, and the output protocol. Although more complicated than the controlled data path, the conversion process is still simple enough to avoid the use of relatively slow microprocessor controlled logic. Within the protocol conversion units, the input data is split into four paths. One path goes to the transparency sub-function shift register, one to the framing sub-function shift register, one to the error control circuit and one to the control code translation shift register.

The framing circuit detects the synchronization character or flag and generates signals to help segment the subsequently received data in word length groups. The transparency circuit detects and strips any stuffed characters or bits in the incoming protocol conversion units. In the output protocol conversion units, the transparency circuit detects any data that should be made transparent to the external communication system and marks the data as such by stuffing characters or bits as appropriate. The error circuit generates or checks the block check characters depending on its use in an input or an output protocol conversion unit.

In operation, the input protocol conversion unit is initially operating in a sync or flag search mode depending

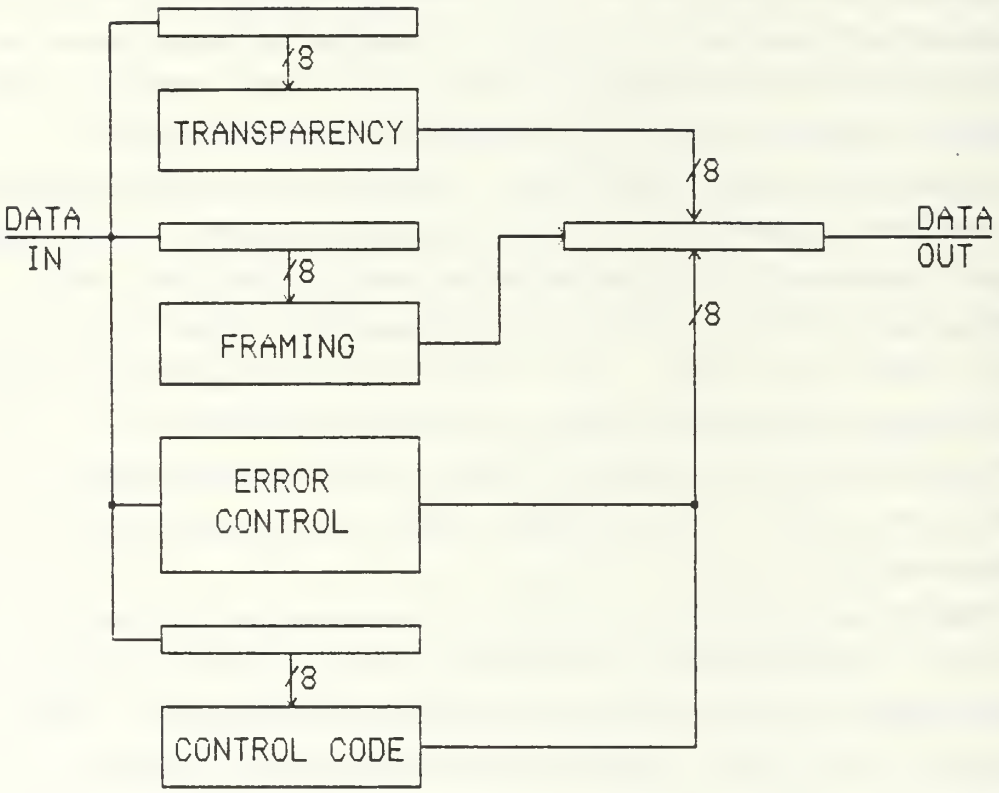

Figure 4.5 Protocol Conversion Unit Block Diagram

on the input protocol selected. The conversion unit checks each word length group of bits of the incoming data stream for the expected sync/flag character. The input data is clocked into a shift register a bit at a time. After each bit is clocked in, one word length of the bits in the shift register are compared with a stored bit image of the sync/flag character. A bit for bit match indicates that the sync/flag character has been received.

Once the sync/flag character is detected, the input protocol conversion unit starts interpreting each successive word length of bits as a single character. This continues until the indicated number of bytes/characters have been received in the case of a byte count protocol, or the ending control character or flag is detected, in the case of character oriented and bit oriented protocols.

Multiple sync/flag characters are typically sent by the transmitting unit to insure synchronization of the framing sub-function, even if one sync is disrupted by noise in the communication channel. Each sync/flag character is detected by the protocol conversion unit, the first one received which is followed by other than another sync/flag character is used for synchronization.

The sync/flag search circuitry of the protocol conversion units consists of a shift register, a storage register to hold a bit image of the expected sync/flag character, and a PLA dedicated to detecting equivalence between the two registers. The sync/flag search circuitry for all three different framing techniques protocols is basically the same with slight variations which are described below.

In the character oriented protocols, a special synchronization character SYN is used to indicate the beginning of a frame. The specific bit sequence for the character SYN is a parameter input which is set before operation. Parameterizing the bit sequence of the SYN

character allows for variations between the different implementations of character oriented protocols.

To implement detectors for all the control characters used in character oriented protocols requires a storage register for each character bit image to be detected. Each bit of each storage register must be compared to its corresponding bit in the first word length of bits of the input shift register with a detect PLA. The output of the detect PLA is used to flag the detection of its associated control character.

Byte count protocols, like character oriented protocols use a special synchronization character SYN to indicate the beginning of a frame. However, unlike character oriented and bit oriented protocols, there is no control character or flag to indicate the end of a frame. The header of a byte count protocol frame includes a byte count indicating how many of the bytes following the header are part of the information field of the frame.

Implementation of the framing sub-function in byte count protocols is the same as for character oriented protocols, except there is only one control character to be detected. The input data stream is clocked into a shift register and compared to a stored bit image of the expected sync character with a detect PLA. Once a sync character is detected, the next byte is interpreted as the class of the message, and the fourteen bits after that are the byte count. The byte count is read and used to set a counter to keep track of the length of the information field of the frame.

Bit oriented protocols also use a special character called a flag to mark the beginning of a frame. The same flag is used to indicate the end of a frame, and consequently it can also be interpreted as the beginning of the next frame. The use of the same flag to indicate the beginning and the end of a frame is very economical in the

use of hardware. There is only one control character or flag to be detected. Only one storage register is required to store the expected flag bit image, and only one detect PLA is required to detect a match with the bits in the shift register.

The bit image of the flag is a parameter input. The parameterization of the bit sequence allows for variations between implementations of bit oriented protocols.

Bit oriented framing is like character oriented framing in that special characters are used to indicate the beginning and end of a frame. The two framing techniques are different in that character oriented framing requires several different control characters. Character oriented framing requires one control character for indicating the start of a frame, one for the start of a frame header, another to indicate the start of the text field, another to mark the end of text field, etc. Bit oriented framing only uses one special character called a flag to delineate a frame. The same character is used to mark the beginning and the end of a frame. This produces fewer control characters to be considered in the transparency sub-function.

The transparency circuit within the input protocol conversion unit strips the stuffed bits and characters from the input data and converts the remaining data to the internal virtual protocol. Before passing the virtual protocol data stream back to the data path controller, an extra control bit, added to each byte, is set to indicate the transparent data to the output protocol convertor. The output protocol conversion unit interprets the control bit of the internal virtual protocol and then converts the remaining data to the output protocol. Bit stuffing or character stuffing is then performed on the data as required by the indicated output protocol.

Character oriented protocols use character stuffing to prevent data from being interpreted as control characters. A control character DLE is reserved for indicating which bit sequences should not be interpreted as control characters, despite their usual control significance. When a DLE character is received, the control character detection mechanisms are turned off while the next eight bits are shifted into the system. That way, the character following a DLE is interpreted as data, no matter what the bit sequence. The stuffed DLE's are stripped out of the data stream by the receiving circuit and are not included in the block check character.

The detection of the control character DLE is accomplished in the same manner as the detection of the other control characters used in framing in character oriented protocols. As the input data stream is stepped into a shift register, one word length of bits is compared with a stored bit image of the DLE character using a detect PLA. The output of the PLA is used to indicate the detection of a DLE and to turn off all control character detection circuits (including itself) during the next eight bit shifts.

Byte count protocols do not use the same circuitry for the transparency sub-function. Once the initial synchronization character is detected, and the number of bytes in the information field read from the header, no more control character detection is required until the frame is completed. A counter is set with the byte count read from the frame header and decremented with each byte that passes through the shift register. Any bit pattern received during the delineated information field is interpreted as other than a control character. The SYN character is the only control character used in byte count protocols, and the detection circuitry for it is turned off for the duration of the reception of the information field.

Bit oriented protocols use bit stuffing to prevent data from being interpreted as the flag character. The typical flag character bit sequence is 01111110. Any sequence of five ones in other than a flag character is separated from subsequent ones with a stuffed zero. The stuffed zero is interpreted as such by the receiving circuitry and stripped out before the bit stream is translated into a character.

In the bit oriented input protocol conversion unit, a detect PLA is used to search for any set of five sequential ones as the data is clocked into a shift register. Another register is used to hold the bit image of five ones, and is compared with the first five bits clocked into the shift register. When five consecutive ones are detected, the following stuffed zero is stripped from the data stream. The bit stripping takes place before the eight bits are compared against the bit image of the synchronization flag. A detect PLA is used instead of random logic to allow for the implementation of various flag characters.

In the bit oriented output protocol conversion unit, another detect PLA is used to search for a set of five consecutive ones in the data stream before it is returned to the data path controller. If five ones are detected, a zero is stuffed into the data stream following them to prevent the five ones and any subsequent ones from being erroneously interpreted as the synchronization flag by the external communication system.

According to McNamara [Ref. 3] Cyclic Redundancy Checks (CRC) are considered to be the most effective means for detecting transmission errors in serial data. CRC use a feedback arrangement to combat the tendency of errors in information transmission systems to occur in burst.

The output of the CRC depend collectively on all the digits received in a single frame. Any single digit of a frame received in error makes the entire frame useless. The arrival of a digit is recorded in the stages of a shift

register and manipulated as subsequent digits are received. Once an entire frame has been received, the status of the shift register segments are used to determine if any of the digits were incorrectly received.

Figure 4.6 illustrates the three parts of the CRC; the message data to be transmitted, the generator polynomial, and the constructed message which is actually sent. The constructed message consists of the desired message data plus a series of M bits called the block check characters. The block check characters are generated by appending M zeros to the message data and dividing the appended message data by the generator polynomial. The division is actually the X-OR function between the K bit generator polynomial and K bit sections of the appended message data. The resulting quotient is discarded and the remainder becomes the block check character. The block check character bits are then added to the message data to form the constructed message.

A CRC generator produces the block check character and appends it to the message data before the constructed message is transmitted. At the receiver, the CRC shift register performs a similar division operation, where the received constructed message is divided by the same generator polynomial. See Figure 4.7. Once the division is completed, if the quotient contains a remainder, there was an error in the constructed message received. That is, the message data or the block check character was received incorrectly, and the message must be discarded.

Several generator polynomials have been accepted as standards for different length words. Table 4.4 list the three most common.

Example Generator Polynomial: 1 0 0 0 1

Example Message Data: 1 0 0 1 0 1 1 0

Appended Message Data: 1 0 0 1 0 1 1 0 0 0 0 0 0

Division of Appended Message Data by the Generator Polynomial:

```
                1 0 0 1 1 1 1 1 1   Quotient (discarded)
1 0 0 0 1 / 1 0 0 1 0 1 1 0 0 0 0 0
            1 0 0 0 1
                1 1 1 1 0
                1 0 0 0 1
                  1 1 1 1 0
                  1 0 0 0 1
                    1 1 1 1 0
                    1 0 0 0 1
                      1 1 1 1 0
                      1 0 0 0 1
                        1 1 1 1 0
                        1 0 0 0 1
Block Check Character:    1 1 1 1
```

Appended Message Data: 1 0 0 1 0 1 1 0 0 0 0 0 0

Block Check Character: + 1 1 1 1

Constructed Message Data: 1 0 0 1 0 1 1 0 0 1 1 1 1

Figure 4.6 CRC Block Check Character Generation

Example Generator Polynomial: 1 0 0 0 1

Constructed Message Data: 1 0 0 1 0 1 1 0 0 1 1 1 1

Division of Constructed Message Data by the Generator Polynomial:

```
                    1 0 0 1 1 1 1 1 1   Quotient (discarded)
1 0 0 0 1 / 1 0 0 1 0 1 1 0 0 1 1 1 1
            1 0 0 0 1
                  1 1 1 1 0
                  1 0 0 0 1
                    1 1 1 1 0
                    1 0 0 0 1
                      1 1 1 1 1
                      1 0 0 0 1
                        1 1 1 0 1
                        1 0 0 0 1
                          1 1 0 0 1
                          1 0 0 0 1
                            1 0 0 0 1
                            1 0 0 0 1
          Remainder:
```

Figure 4.7 CRC Error Detection

TABLE 4.4 COMMON CRC GENERATOR POLYNOMIALS

$$\text{CRC-12} = X^{12} + X^{11} + X^3 + X^2 + X^1 + 1$$
$$\text{CRC-16} = X^{16} + X^{15} + X^2 + 1$$
$$\text{CRC-CCITT} = X^{16} + X^{12} + X^5 + 1$$

The CRC-12 is used with 6-bit characters, the CRC-CCITT and CRC-16 are used with 8-bit character systems. The length of the polynomial is the same as the length of the burst of errors that it can detect with 100% assurance. Any burst of errors longer than the polynomial can be detected with a 99.9% assurance. A sixteen bit checksum such as CRC-16 or CRC-CCITT will detect all single and double errors, and all errors with an odd number of bits. [Ref. 2:p. 132]

Most of the currently popular protocols require the initialization of the CRC shift register to zero before shifting the data through. Two exceptions to this procedure are the SDLC and HDLC protocols. They both require a preset value in the shift register segments of one. Once the frame is shifted through, the indication of no errors in the transmission and reception of the frame is a special nonzero result in the shift register segments. The initialization of the CRC shift register is one of the parameter inputs, with the default value being all zeros.

There are slight but significant differences between the conversion unit circuits depending on their use. These variations are required by the differences between the three framing technique protocols.

The framing circuit of the input protocol conversion unit detects the synchronization character for the character oriented and byte count protocols, and the synchronization flag for the bit oriented protocols. After detecting the synchronization character or flag, the framing circuit inserts a copy of the virtual protocol flag in the data sequence.

The input protocol conversion unit for the character oriented protocols detects the single occurrence per frame of the characters SYN, ETX and ETB. If there is a double occurrence and detection of the SYN character, the last SYN to arrive is used for synchronization. The input protocol conversion unit for the byte count protocols detects the occurrence of the SYN character only. The input protocol conversion unit for the bit oriented protocols detects the occurrence of the synchronization flag. The synchronization flag occurs twice per frame in bit oriented protocols. The framing circuit for the bit oriented protocols must remember the first occurrence of the synchronization flag to interpret the second occurrence as the end of the frame flag.

The transparency circuit of the input protocol conversion unit detects and strips stuffed characters or bits in the character oriented and bit oriented input protocol conversion units. In the byte count input protocol conversion unit, the byte count in the frame header is determined by the transparency circuit. A count is kept of the subsequent bytes that pass through the transparency circuit shift register, and a virtual protocol flag is appended to the frame once the prescribed number of bytes have passed through.

The error control circuit of the input protocol conversion unit checks the block check characters of the incoming frame, for all three types of protocols. The generator polynomial used is typically different for the three types of protocols. The location of the input protocol conversion unit relative to the transmitting station determines the requirement for setting a control bit if errors are detected. If the unit is located adjacent to the station with a minimum of error susceptible channel between them, no indication of errors is necessary from the input protocol conversion unit.

The control code translation circuit of the input protocol conversion unit converts the information control codes of the input protocol to a generic set of codes used in the virtual protocol. The three types of protocols use different control codes for transfer of the same control information, requiring a separate translation table for each protocol type.

The framing circuit of the output protocol conversion units detects the virtual protocol flag inserted by the input protocol conversion unit and replaces it with the synchronization character or flag required by the specified output protocol.

The transparency circuit of the output protocol conversion unit for the character oriented protocols detects the occurrence of any control code bit sequence in the text field of the frame. A DLE character is then inserted before any control code sequences occurring in the text field of the frame by the output protocol conversion unit. The transparency circuit of the output protocol conversion unit for the bit oriented protocols detects the occurrence of five consecutive ones in the data stream to be returned to the data path controller and stuffs a zero into the data stream immediately following the five ones.

The error control circuit of the output protocol conversion unit for all three types of protocols generates the block check characters required by the specific protocol and appends them to the message data. An indication from the input protocol conversion unit of an error in the input data in the form of a set control bit causes the error control circuit of the output protocol conversion unit to invert the bits of the block check character. Inverting the bits of the block check characters virtually guarantees a subsequent error indication by the receiving station of the external communication system.

The control code translator circuit of the output protocol conversion unit detects the generic information control codes of the virtual protocol inserted by the input protocol conversion unit. The virtual protocol control codes are converted to the control codes of the desired output protocol through the use of a translation table.

E. COMMON CIRCUITS

The data path controller circuit is a simple circuit for directing the flow of data within the protocol convertor. The protocol select inputs are interpreted with a decoder and use to control four multiplexers. See Figure 4.8. The first pair of multiplexers direct the input data sequence to the desired input protocol conversion unit, and select the return line from the same input protocol conversion unit to return the results to the data path controller. The second pair of multiplexers determine which output protocol conversion unit will be used, and select the return line from the same output protocol conversion unit to return the manipulated data sequence to the data path controller. The desired data path is determined by the selection of transistor switches which are opened and closed according to the protocol select inputs.

Several circuits are common to both input and output protocol conversion units, in all three framing technique categories. The detection function provided by the detect PLA is required in each input and output protocol conversion unit. In keeping with the concept of the protocol convertor as a filter operating at sufficient speeds to avoid becoming a bottleneck in the system, an optimum design is required. Optimizing the most prevalent circuit with regard to minimum clock period and minimum surface area should ultimately produce a smaller, faster protocol convertor.

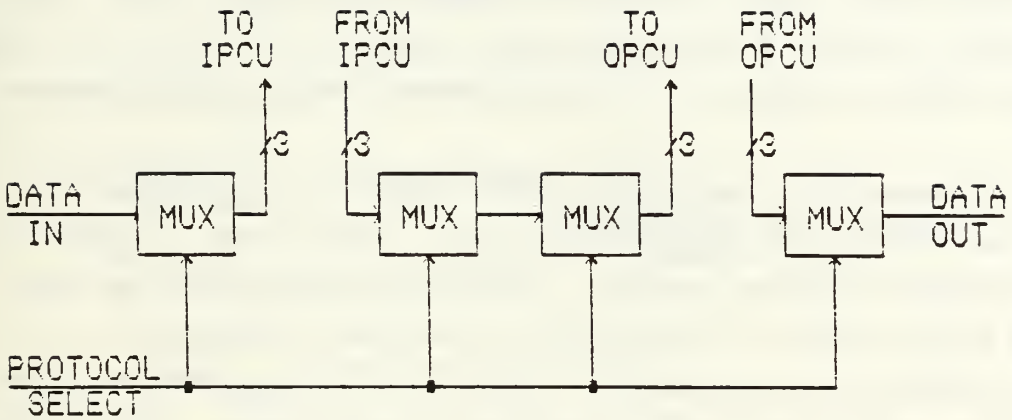

Figure 4.8 Data Path Controller Block Diagram

The character oriented protocol conversion units utilize the detect PLA circuit the most. The character oriented input protocol conversion units use it for searching for the synchronization bit sequence SYN and the transparent data marker DLE in the input protocol conversion units. The character oriented output protocol conversion units also use the detect PLA to search for data sequences that resemble control characters.

The detect PLA is used in the bit oriented input and output protocol conversion units to search for the synchronization flag and any set of five consecutive ones. The byte count input protocol conversion units, like the character oriented input protocol conversion units, use the detect PLA to search for the synchronization bit sequence SYN.

In operation, the detect PLA compares two different data sequences and gives a positive indication if there is equivalence between the two. Each individual bit of the input data stream is combined in the PLA with the corresponding bit of the bit image of the character to be detected using the X-NOR function. See Figure 4.9. If the two bits are identical, the X-NOR function produces a high output, otherwise it will be low. Within the PLA, the output of the eight X-NOR functions are combined together with an AND function. The output of the AND function indicates if the expected sync/flag character was received. If all the bits in one word length of the shift register match the bits of the stored sync/flag character, each of the X-NOR functions will output a high value to the input of the AND function. The subsequent high output of the AND function indicates the sync/flag character has been received. Any bit in the shift register which does not match the corresponding bit of the stored sync/flag character will cause a low value to be sent from the X-NOR function involved to the AND function. A low output of the

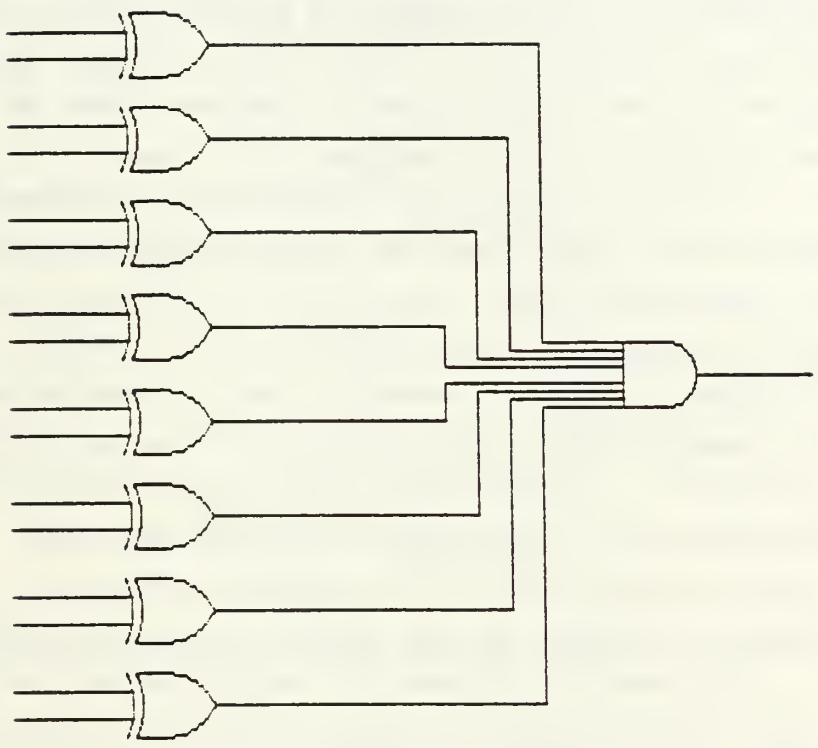

Figure 4.9 Detect PLA Functional Equivalent

AND function indicates that least one bit of the current set of bits in the shift register does not match the bit image of the expected sync/flag character.

PLAs are preferred over random logic in circuit design for several reasons. Using a PLA allows for the implementation of changes in the logic of the circuit, without requiring the redesign of the entire system. The shape and size of a PLA is dependent only on the number of inputs and outputs and the number of product terms required to implement the desired function, so any changes in the logic required of the circuit can be accomplished by just redesigning the PLA. If the system is designed with some slack as to the area occupied by a PLA, minor changes can be absorbed without effecting the rest of the system.

In some implementations, the use of a PLA can increase the speed of operation of the circuit. The minimum clock period of a circuit is determined by the longest signal path. The longest signal path should produce the longest delay in a circuit. If by using a PLA the longest signal path is shortened, the clock period can be reduced. A shorter clock period equates to a faster circuit.

One drawback of PLAs is the number of transistors required to implement the circuit. Depending on the function implemented with the PLA, the use of a PLA may require more transistors for the overall circuit, even though the longest signal path is shorter. An increase in the number of transistors requires more area on a chip for the circuit.

To determine the best method of implementing a circuit, the advantages and disadvantages of using PLAs must be weighed against each other, for the particular logic function desired. In designing the detect PLA the speed and area factors of a PLA as compared to a random logic version of the same circuit were considered.

Four different PLAs were compared with each other and the random logic equivalent. The random logic equivalent was used as a standard for comparison. The clocked input driving circuitry, which is similar for each, was not considered for either the PLAs or the random logic circuit.

To implement the detect circuit with random logic requires three NOR gates and two inverters for each pair of bits to be compared. One multiple input AND gate is required to collect the NOR gate outputs. See Figure 4.10 and Figure 4.11.

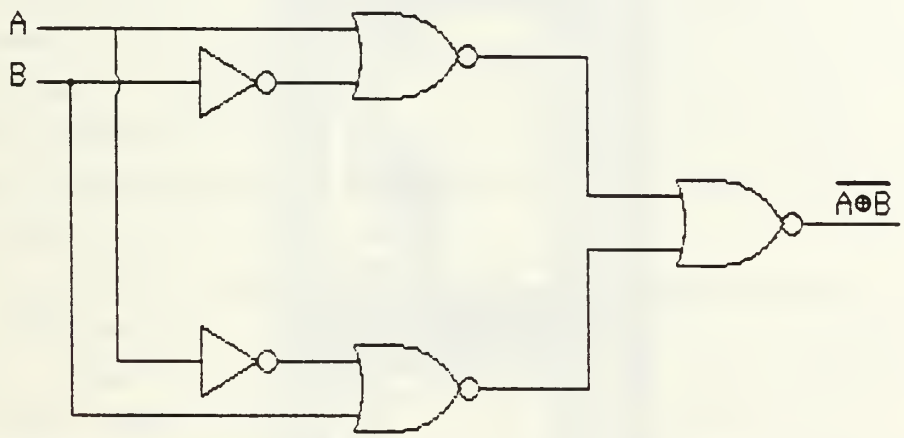

Figure 4.10 Random Logic Detect Circuit for Two Bits

The NOR gates, and the inverters requires two transistors each for a total of ten transistors to implement the X-NOR function between two bits. The multiple input AND gate requires a transistor for each input and one more for the pull-up transistor. A sixteen input detector circuit which would detect equivalence between two sets of eight inputs would require a eighty transistors for the X-NOR function and nine for the AND function, for a total of eighty-nine transistors. The longest signal path is eight transistors.

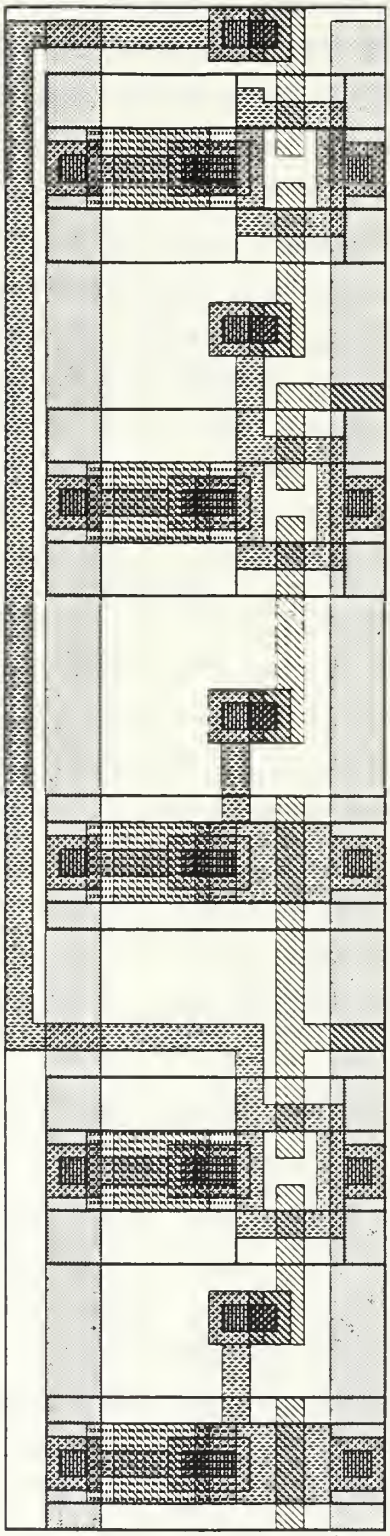

Figure 4.11 VLSI Implementation of a Random Logic Detect Circuit for Two Bits

There are many different ways of implementing the detect circuit with a PLA. The equivalence function can be partitioned into any combination of pairs of bits. Each bit pair can be tested for equivalence independently of all the other bit pairs. The only requirement for the grouping of the inputs is that any two bits to be compared must both use the same PLA.

The detect circuit calls for the comparison of eight pairs of bits which can be accomplished in four symmetric methods. The first method is a single PLA with eight pairs of inputs and one output. The second method is two PLAs with four pairs of inputs each and an AND gate to tie together their outputs. The third method is four PLAs with two pairs of inputs each and a four input AND gate. The fourth method is eight PLAs with one pair of inputs, and an eight input AND gate. The single pair input PLA is included only for completeness.

The longest input path in the PLA implementation of the the X-NOR and AND functions is a function of the number of inputs. There is one more transistor in the longest signal path than there are inputs to the PLA. See Table 4.5. The longest signal path for a single bit in the random logic implementation of the circuit is eight transistors. The number of transistors in the longest signal path is only part of the consideration. What type of gates those transistors compose, and how many of the inputs to those gates are active at one time also has an effect on the total time required to transverse the longest signal path.

The length of the longest signal path can be measured in single inverter delay units or channel transit time tau. For small V_{ds}:

$$tau = L^2/U*V_{ds}$$

TABLE 4.2 PLA SIGNAL PATH LENGTH AS COMPARED TO RANDOM LOGIC

PLA Inputs	Number of Terms	Number Required	Longest Path Internal	tau NAND	Longest Path Total	Percent Increase
16	256	1	17	0	17	53
8	64	2	9	2	11	27
4	16	4	5	4	9	11
2	4	8	3	8	11	27

Where L is the length of the gate in centimeters, U is the mobility of the electrons in centimeters squared per volt-second, and V_{ds} is the voltage difference between the drain and the source of the transistor in volts. [Ref. 20]

The NOR gate is the principal building block of the random logic detect circuit, and all of the NOR gates used have two inputs. The delay induced by a NOR gate is a function of the number of active inputs. The detect circuit is used primarily to detect the synchronization characters or flags, which occur on the average of twice per frame; twice at the beginning, or once at the beginning and once at the end. If the average total number of bytes in the frame is much greater than two, ninety-nine percent of the time the circuit will be indicating no equivalence. This means at least one of the eight pairs of bits will not be the same ninety-nine percent of the time. Considering an equal probability of receiving a one or a zero at any time, there is a fifty percent probability of the two inputs to a NOR gate being the same and a fifty percent probability of the two inputs being different. If the two inputs are both low, there is no delay experienced by the signal. This occurs twenty-five percent of the time. If one of the inputs is low, and the other one is high, the signal will experience a single tau delay. This occurs fifty percent of the time. If both of the inputs are high, the signal experiences a delay less than tau. This also occurs twenty-five percent of the time. Therefore, the signal will be experiencing a single tau delay or less through each NOR gate of the circuit.

The delay imposed by a NAND gate is proportional to the number of inputs, regardless of the state of the inputs. As the number of inputs to a NAND gate is increased the length of the pull-up transistor area must be increased proportionally. As the area of the pull-up transistor is

increased, the amount of delay imposed on the signal is also increased.

$$tau_{NAND} = n*tau$$

Where n is the number of inputs.

Using more PLAs of fewer inputs reduces the length of the longest signal path internal to the PLA. But, with the addition of another PLA comes the requirement for an NAND gate to combine the outputs of the multiple PLAs. The delay imposed on the signal by the NAND gate is proportional to the number of inputs, causing the total delay to actually increase as the number of PLAs is increased past four. The delay experienced by the signal through PLA is shorter when more PLAs of fewer inputs are used, but the additional delay imposed by the requisite NAND gates negates any gains achieved by using more than four PLAs.

A unique minimum could be determined if the equivalence function could be partitioned into continuous numbers of inputs. However, the discrete partitioning of the equivalence function produces numbers of inputs of powers of two only. Table 4.5 indicates a local minimum at four PLAs with two pairs of inputs each.

The number of transistors used in the implementation of the detect circuit with a PLA is a function of the number of inputs. If k is the number of inputs into a single PLA, the total number of transistors T required to implement the logic of the PLA alone, not including any input or output drivers is:

$$T = (k + 1)*2^{k/2}$$

This relationship imposed on the discrete numbers of inputs allowed by the partitioning of the equivalence function is

presented in Table 4.6. The random logic implementation of the detect circuit requires eighty-nine transistors.

Related to the number of transistors in each PLA is the area the PLA occupies on the chip. Circuit areas are typically measured in square length units. The length unit is defined as the fundamental resolution of the fabrication process.

> . . . (the length unit) is the distance by which a geometrical feature on any one layer or on another layer may stray from another geometrical feature on the same layer or on another layer, all processing factors considered and an appropriate safety factor added [Ref. 20:p. 48].

The areas of the different PLA implementations are presented in Table 4.7. The area of the random logic implementation of the detect circuit is approximately thirty-six thousand square length units.

When considering speed of operation and the area occupied by the circuit, four PLAs with two pairs of inputs each produce the optimum design. Although eleven percent slower than the random logic design, the four PLAs occupy approximately two thirds of the area, and use almost exactly the same number of total transistors. The design flexibility incurred by using PLAs is another factor in their favor.

Each output protocol conversion unit requires a Cyclic Redundancy Checks (CRC) shift register to generate block check characters. The input protocol conversion units also require CRC shift registers if the protocol convertor is to be located anywhere other than physically adjacent to the transmitting station of the external communication system. The input protocol conversion units use the CRC shift registers to check the input data for errors.

The CRC shift register circuit is the same for both the input and the output protocol conversion units. Error detection is accomplished with the same basic circuit as block check character generation. There are some

TABLE 4.3 NUMBER OF PLA TRANSISTORS AS COMPARED TO RANDOM LOGIC

PLA Inputs	Number of Transistors	Number Required	AND Gate Transistors	Total Transistors	Difference Factor
16	4352	1	0	4352	48.90
8	144	2	2	290	3.26
4	20	4	4	84	.94
2	6	8	8	56	.63

TABLE 4.4 AREA OF PLA CIRCUITS AS COMPARED TO RANDOM LOGIC

PLA Inputs	Length	Width	Area	Number Required	Total Area	Difference Factor
16	323	2127	687021	1	687021	9.54
8	195	143	27885	2	55770	.77
4	131	47	6157	4	24628	.34
2	99	31	3069	8	24552	.34

peripheral differences required by the difference in the detection function and the generation function.

A CRC shift register is a shift register with an X-OR gate inserted between each stage of the shift register. See Figure 4.12 and Figure 4.13.

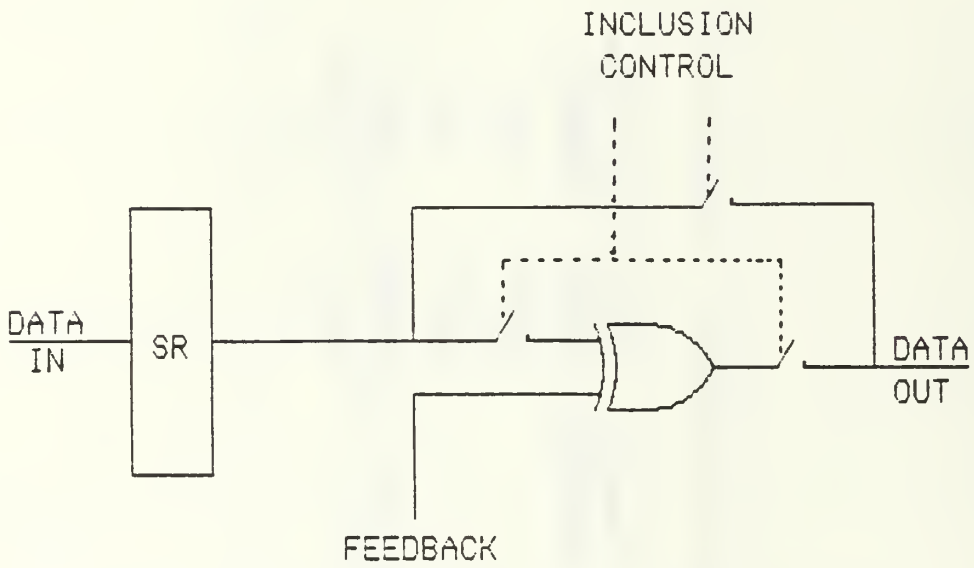

Figure 4.12 CRC Shift Register Segment

As bits are clocked into the shift register, they pass through an X-OR gate before arriving at the subsequent stage of the shift register. The other input to each X-OR gate is a feedback term from the input data. The output of the X-OR gates are fed into the next stage of the shift register.

The configuration of a CRC shift register is dependent on the generator polynomial. The number of shift register stages is equal to the degree of the generator polynomial. The number of X-OR gates connecting the feedback line to the shift register segments is equal to the number of terms in the generator polynomial. If a term is included in the generator polynomial, the output of that shift register segment is combined with the feedback term and passed

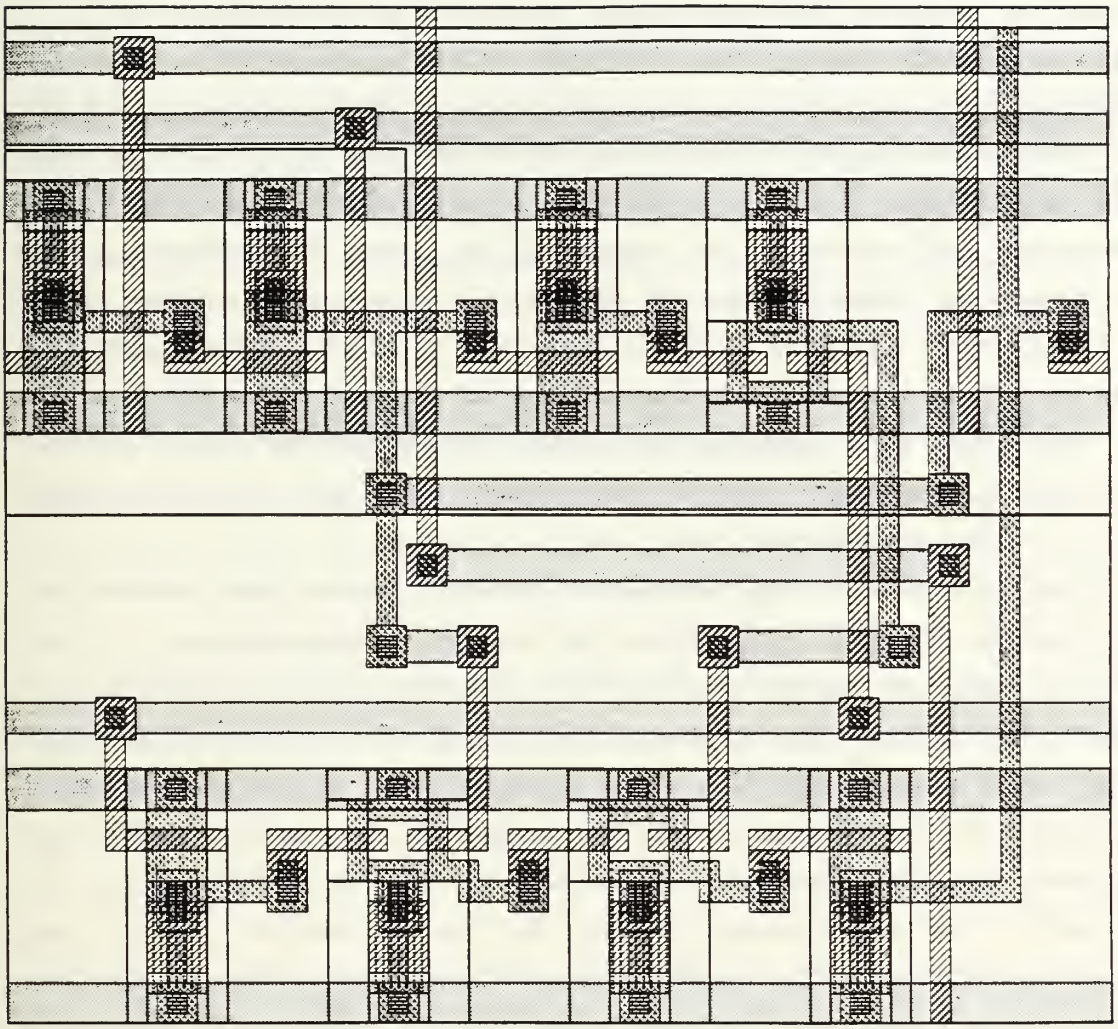

Figure 4.13 VLSI Implementation of a CRC Shift Register Segment

through an X-OR gate into the input of the next shift register segment. If the term is not to be included, the shift register segment output is connected to the input of the next shift register segment without any effect by the feedback term.

In a parameterized protocol convertor, any combination of terms for the CRC generator polynomial should be available. To accomplish this, all of the shift register segments are equipped with X-OR gates. The inclusion of the X-OR'd feedback terms is controlled through the use of pass transistors. If a term is to be included in the generator polynomial, the controlling pass transistors are set such that the output of the shift register segment goes through the X-OR gate and is combined with the feedback term. Otherwise the controlling pass transistors bypass the the X-OR gate completely, and the output of the shift register segment is fed directly into the input of the next shift register segment.

The first and last terms of the generator polynomial are always included, so there is no pass transistor controlling the path of the first shift register segment output. The last term, even though it is always included, still uses the controlling pass transistor to allow for a variable CRC word length. The length of the CRC shift register is a function of the system word length; a parameter input.

The input protocol conversion units uses the CRC shift register for error detection. Once the input data stream has been shifted through the CRC shift register, the shift register segments are tested to determine if any errors were detected in the incoming data stream. In the case of character oriented or byte count input protocol conversion units, this is accomplished by combining the status of all the shift register segments into a NOR gate. If the output of the NOR gate is one, there were no errors. A zero

output from the NOR gate indicates the frame or the block check character were received with errors. Bit oriented input protocol conversion units require the shift register segments be tested against a stored bit image to determine if any bits were received in error. This is accomplished using a detect PLA.

The process of passing a frame through a CRC shift register destroys the information content of the frame. This requires the parallel input of the received frame into both a CRC shift register for error detection and other shift registers for other protocol sub-function manipulations. The parallel input is accomplished by copying the input data stream into multiple shift registers simultaneously.

The output protocol conversion units use the CRC shift register for block check character generation. As the converted data stream is output back to the data path controller, the bits are copied into the CRC shift register. When the second virtual flag is received by the output protocol conversion unit, indicating the end of the message data, the contents of the CRC shift register are appended to the converted data stream. Once all the bits have been clocked through the CRC shift register, the shift register contents are the block check characters.

Each of the different framing techniques relies on a set of bit sequences called control characters to accommodate the required sub-functions of operation. Bit oriented protocols use the synchronization flag, and byte count protocols use the SYN character for synchronization. The character oriented protocols by their design use the most control characters. Depending on their use, some control characters require activity on the part of the protocol convertor, others can be passed on to the external communication system as information. Those control characters that require activity by the protocol convertor

must be detected and a response initiated. These include the character oriented protocol transparent data indicator character DLE, and the synchronization character SYN. Those control characters that do not require activity by the protocol convertor are converted to a set of virtual protocol control characters with a translation table and passed to the output protocol conversion unit. There the virtual protocol control characters are converted back into those control characters expected by the receiving station. Examples of information conveying characters which do not require activity on the part of the protocol convertor are the character oriented protocol ACK acknowledge, and ENQ enquiry. In keeping with the concept of a flexible protocol convertor, the translation tables should be stored in an accessible medium.

F. SUMMARY

A typical example of protocol conversion was described to illustrate the basic requirements of the protocol conversion process. Examples of the frame formats, parameter inputs and the control code translation table for two different protocols were followed by a frame-by-frame description of the conversion process.

Next, a description of the convertor architecture required to implement a fast, yet flexible protocol conversion process was presented. The top-level description included the three different types of inputs into the system and the data path through the system.

The top-level architecture description was followed by an in-depth description of the protocol conversion process using only hardware. The central concept of the conversion process is a virtual protocol used internally to the protocol convertor. Selection of the components of the virtual protocol is the key to simplifying the conversion process.

Next follows an analysis of the protocol sub-functions required of the protocol convertor; framing, transparency, and error control. Framing is required for the recognition of and establishment of frame boundaries. Transparency permits the transmission of data that would otherwise be interpreted as a control character. Error control generates block check characters and checks input data for transmission errors. The inclusion of the sub-functions of line control, time-out control, sequence control and initialization in the protocol convertor is unnecessary The implementation of these functions by the external communication system is sufficient.

The description of the required sub-functions is followed by a description of the circuits used to implement them. The framing circuit detects the synchronization character or flag and generates signals to help segment the subsequently received data in word length groups. The transparency circuit detects and strips any stuffed characters or bits in the incoming protocol conversion units. In the output protocol conversion units, the transparency circuit detects any data that should be made transparent to the external communication system and marks the data as such by stuffing characters or bits as appropriate. The error circuit generates or checks the block check characters depending on its use in an input or an output protocol conversion unit.

Two circuits of the protocol convertor are described in detail, the detect PLA and the CRC Shift Register. In keeping with the concept of the protocol convertor as a filter operating at sufficient speeds to avoid becoming a bottleneck in the system, an optimum design is provided for the most prevalent circuit, the detect PLA. The error control functions provided by the CRC Shift Register are usually implemented with software and the CRC Shift Register is a hardware implementation of the CRC algorithm.

V. IMPLEMENTATION

A. CHIP DESIGN

Several relatively large circuits are required in each protocol conversion unit, such as the detect PLA. The standard 40 pin VLSI package can support a silicon chip of approximately 7 millimeter square dimensions [Ref. 20:p. 131]. Using 2.5 micrometer unit length technology, 7 square millimeters equates to 2800 square lambda for the entire protocol convertor.

The surface area available for the protocol convertor circuit is reduced somewhat from the 2800 square lambda by the area required for input and output pads. Using the typical pad dimension of 100x150 lambda [Ref. 21], and placing 10 pads on each side of the chip reduces the available area to approximately 2500 square lambda.

Newkirk and Mathews [Ref. 21] list a shift register design which occupies 24x90 lambda per bit. See Figure 5.1.

Figure 5.1 Shift Register Segment

The shift register requires a two-phase nonoverlapping clock (PHI1 and PHI2) for storage and shifting of the data through the register. Using this design produces a 192x190 lambda design for an eight bit shift register.

The detect PLA with two pairs of input described in Chapter Four occupies 195x143 lambda. See Figure 5.2.

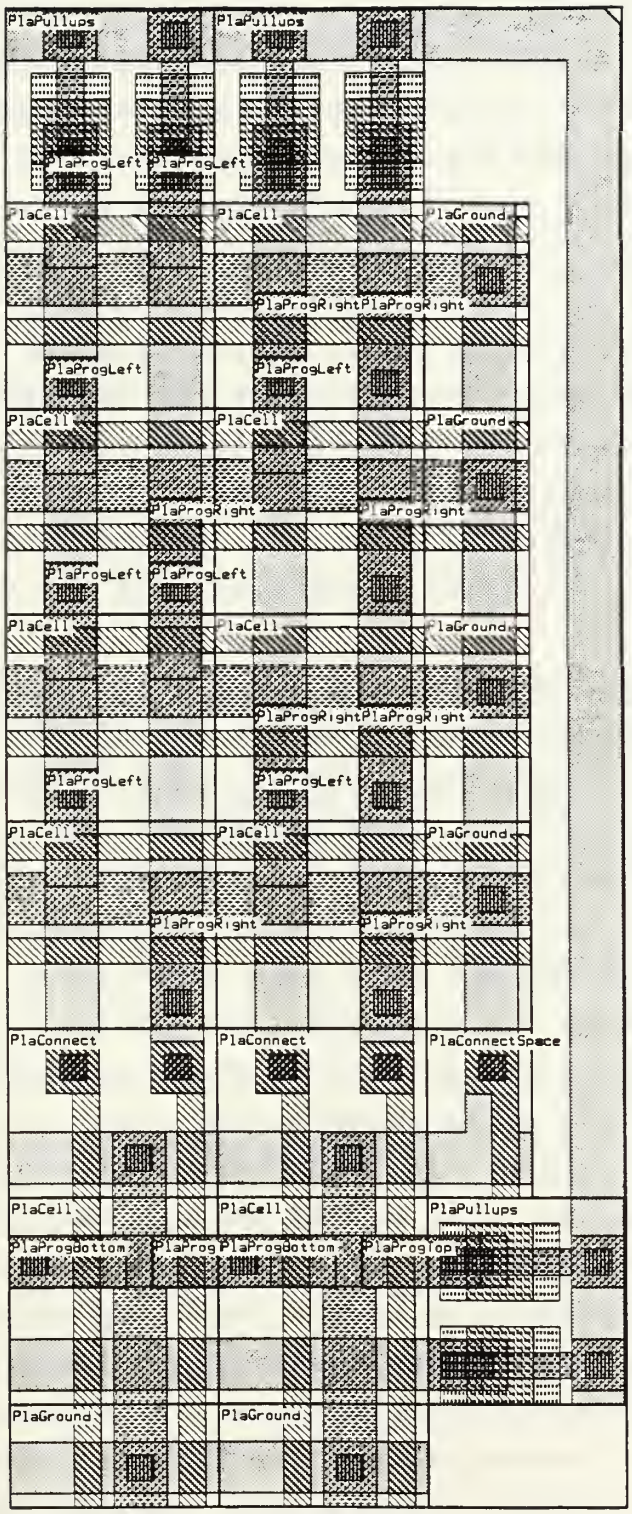

Figure 5.2 A Two Pair Input PLA

Four PLAs with two pairs of inputs each are required to detect the equivalence between eight pairs of bits for a total area of 780x195 lambda for each detect PLA.

The custom designed CRC shift register requires an area of 90x110 lambda per bit. See Figure 4.13. A sixteen bit CRC shift register can be constructed as two vertically stacked eight bit CRC shift registers for a total area of 720x220 lambda.

The major consideration in determining the layout of the chip is the central location of the data path controller. See Figure 5.3. All input signals must pass through the data path controller twice. Centrally locating the data path controller makes the average path to both the input and output protocol conversion unit the shortest possible.

The input and output signals can be bussed through the protocol conversion units or routed in the 200 lambda wide channels between the units. Bussing the signals through the protocol conversion units places routing constraints on the V_{dd} and ground net. There is sufficient area on the chip to utilize channels between the different protocol conversion units and the data path controller. The input and output signals run horizontally; parallel to the V_{dd}, ground and clock signals. All of these signals are implemented on the metal layer of the chip.

The control signals for the different protocol conversion units are also routed through the 200 lambda wide channels between the units. These signals run vertically, perpendicular to the input and output signals. The control signals are implemented in polysilicon.

The system clock is dependent on the signalling speed of the external communication system. The modem which demodulates the digital information from the analog signal also extracts a clock signal from the analog signal. The clock signal is input into the clock input of the protocol

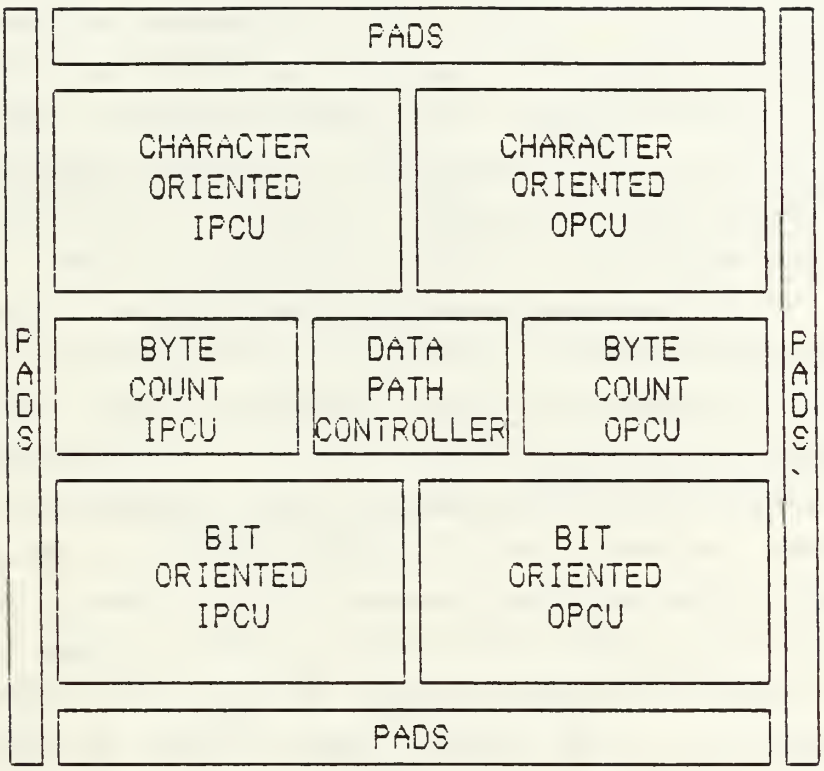

Figure 5.3 Chip Floor Plan

convertor and controls the timing of the system. The modem extracts a single phase clock signal from the input data, which is converted into a two-phase, nonoverlapping clock signal within the protocol convertor. The two-phase clock signal is generated with the clock pad circuit described by Newkirk and Mathews [Ref. 21:p. 111]. See Figure 5.4.

The predicted signal delay is a function of the longest shift register used in the conversion process. The majority of the protocol conversion process is accomplished on a single byte at a time. It is necessary to store one byte within the protocol convertor to detect bit sequences dependent on the adjacent relationship between eight consecutive bits. With one byte in storage within the protocol convertor, the overall transfer of information is going to be delayed by one byte of transfer time. For example, if the external communication system is exchanging information at a rate of 1200 bits per second, or a period of 0.833 milliseconds per bit, a delay of eight bits would equate to a total transfer delay of 6.67 milliseconds. The transfer delay incurred by passing the information through the protocol convertor would not be evident to the user on either end of the external communication system. The transmitted data would just arrive eight bit times later than if it had not passed through the protocol convertor.

In contrast, if the entire frame of data is stored in the protocol convertor, the delay would be noticeable. The frame would have to be stored, manipulated, and then passed on to the external communication system, A byte count protocol frame can contain up to a maximum of 16373 bytes of data [Ref. 3:p. 158]. Using the same 1200 bits per second information exchange rate, it would take 110 seconds to receive and store the 130984 bits of the frame. If the same parallel processing techniques were used the actual processing time would be equivalent to the single byte storage method. Ignoring the equivalent processing times,

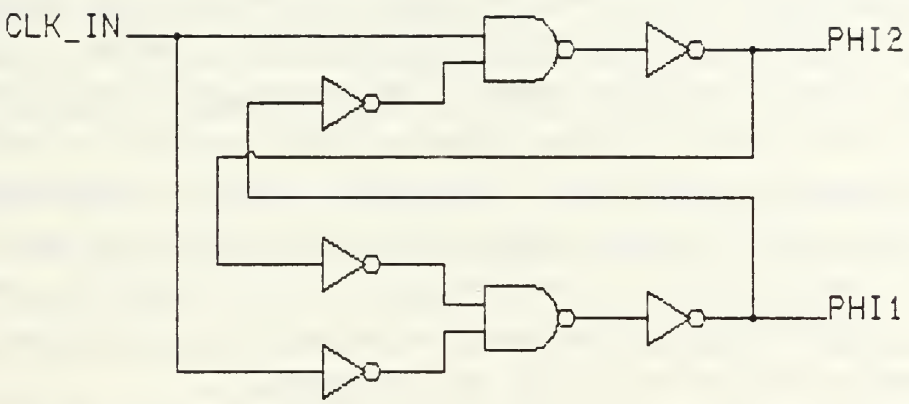

Figure 5.4 Clock Pad Circuit

it would take another 110 seconds to put the converted frame back on the communication channel at the same 1200 bits per second. The entire process, not counting the processing time, would take 220 seconds; a factor of $33x10^3$ slower than the single byte storage method. This is a worst case analysis, using the longest frame expected. If the average frame was just one half of the maximum possible length, the total frame storage technique would still be a factor of $16.5x10^3$ slower.

The success of the single byte storage method of protocol conversion depends on the minimum clock period of the circuit. The single byte storage method processes the frame one byte at a time, without any processing delays other than the initial shift register fill delay. The entire circuit must be capable of operating with a clock period less than the time required for one bit to shift into the shift register. The entire parallel processing of the byte of data held in the shift register must take place before the next bit is shifted in. An information transfer rate of 1200 bits per second corresponds to a minimum processing period of 0.833 milliseconds.

The circuits of the protocol convertor all operate in parallel, so the delay through them is not cumulative. The overall delay of the protocol convertor will be the longest delay of any one of the circuits. The delay of each circuit is independent on any delay in the other circuits. The two circuits of primary concern in delay estimation are the two circuits with the longest signal paths; the CRC shift register circuit and the detect PLA circuit.

The two basic building blocks of the CRC shift register circuit, inverters and NOR gates, both produce the same delay if only one of the NOR gates is active. This is the worst case condition, as the delay for the NOR gate decreases with an increase in the number of active gates.

The inverter Z ratio of pull-up transistor area to pull-down transistor area is five to one throughout. This is slightly more than the Mead and Conway [Ref. 20] recommended four to one ratio to compensate for the pass transistors between the inverters used in the shift register segments. There is a maximum of one pass transistor between any two inverters, so no level restoration is needed.

The two clock phases of the circuit need to be of different duration because of the X-OR logic circuit between the output of one shift register segment and the input of the next. Although this logic is only included in the signal path as a function of the variable generator polynomial, it must be considered present in between all shift register segments when considering worst case timing delays.

Phase One of the clock permits data to be stored on the first inverter of the shift register segment inverter pair. See Figure 4.13. There are no logic functions between the two inverters of the shift register segment, so the duration of Phase One can be minimal. Since stray capacitance delays are at least equal to the circuit delay, the total delay for Phase One will be:

$$Phi_1 = 2*k*tau = 10*tau$$

Where k is the ratio of pull-up transistor area to pull-down transistor area and tau is the technology dependent unit delay.

Phase Two of the clock permits the data signal to pass through the second inverter of the shift register segment. This signal then passes through the X-OR logic if that particular term of the generator polynomial is to be included, or bypasses the X-OR logic if the term is not to be included. The longest signal path through the X-OR

logic is four gates. Doubling the delay for stray capacitance, the minimal length of Phase Two must be:

$$Phi_2 = 8*k*tau = 40*tau$$

Where k is the ratio of pull-up transistor area to pull-down transistor area and tau is the technology dependent unit delay.

Path delays are minimal because of the relative proximity of connected circuits, and long runs between circuits are done in metal. They are approximated as 9 tau for path delay and another 8 tau for driver delay.

All total, Phase One should be approximately 30 tau, and Phase Two should be approximately 60 tau. The total delay of the circuit requires a minimum of a 90 tau clock period. With tau equal to approximately 0.3 nanosecond, this circuit could operate with approximately a 30 nanosecond clocking period, or at a frequency of 33 megahertz.

The internal signal path of the detect PLA is composed of single input NOR gates with a Z ratio of 4. Phase One of the clock can again be minimal because of the lack of combinational logic functions between the input shift register and the PLA:

$$Phi_1 = 2*k*tau = 8*tau$$

Where k is the ratio of pull-up transistor area to pull-down transistor area and tau is the technology dependent unit delay.

The longest signal path through the four input PLA is 5 tau which is added to the external NAND gate signal path of 4 tau for a total signal path delay of 9 tau. See Table 4.2. Again doubling the value to account for stray

capacitance delays, the total minimum period of Phase Two is:

$$Phi_2 = 18*k*tau = 72*tau$$

Where k is the ration of pull-up transistor area to pull-down transistor area and tau is the technology dependent unit delay.

Path delays are again approximated as 9 tau and driver delays as 8 tau. All total, Phase One should be approximately 25 tau and Phase Two should be 90 tau. The total delay of the circuit requires a minimum of a 115 tau clock period. With a tau equal to approximately 0.3 nanoseconds, the detect PLA can operate with approximately a 34.5 nanosecond clocking period, or at a frequency of 29 megahertz.

Both the CRC shift register circuit and the detect PLA circuit are capable of operating with a sufficiently short clock period to permit the use of the single byte storage method of protocol conversion.

B. SYSTEM DESIGN

There are two alternatives to implementation of a parameterized protocol convertor within a communication system. The protocol convertor can be installed internally to the communication system hardware, or externally in its own system environment.

The internal implementation of the parameterized protocol convertor is recommended for several reasons:

- Simplification of the error control sub-function.
- To minimize redundant hardware.

The error control sub-function of the protocol conversion units would not be required to perform error detection if the protocol convertor is located adjacent to the transmitting unit of the external communication

system. Locating the protocol convertor internal to the transmitting station of the external communication system would guarantee a noise free channel between the transmitting station and the protocol convertor. The data received into the protocol convertor could be assumed to be error free and the only error control functions required would be block check character generation by the output protocol conversion units.

Locating the protocol convertor internally to the communication system also reduces the number of MODEMs and serial interfaces required to interface the convertor to the communication system. The protocol convertor expects input signals in the same format as those sent to a serial interface by a central processing unit. The output of the protocol convertor is in the same format as the data a central processing unit places on its busses to be sent to a serial interface. If the protocol convertor is located internal to the communication system, it can be spliced in between the serial interface and the output port of the central processing unit. Locating the protocol convertor within the communicating system allows the protocol convertor to utilize the services of the serial interface and MODEM already in place. This reduces the amount of hardware required to integrate the protocol convertor into the communication system.

Two alternatives are possible if the protocol convertor is to be installed as a separate unit within the communication system. The choice of which implementation depends on the intensity of expected traffic through the protocol convertor.

If a continuous traffic load is expected, a fast, but hardware intensive implementation should be used. This implementation requires two half-duplex MODEMs and two serial interfaces per protocol convertor, or one full-duplex MODEM and two serial interfaces per protocol

convertor. The design of the protocol convertor as a filter with minimal storage implies an ability to pass information through the convertor as fast as it is received. Manipulating data without storage requires a MODEM channel and a serial interface to receive the input data stream, and another MODEM channel and serial interface pair to transmit the output data stream at the same time. A single MODEM channel, serial interface pair would not be sufficient. Half-duplex MODEMs can only transmit or receive at one time. Two of these would be required to receive the input data and transmit the output data simultaneously. A full-duplex MODEM can both transmit and receive at the same time, but only if the two data sequences are separated in the frequency spectrum. One full-duplex MODEM would be sufficient, if each MODEM channel was equipped with its own serial interface.

In a less intensive traffic load environment, a slower, but less expensive implementation could be used. A single half-duplex MODEM and a single serial interface could be used between the protocol convertor and the communication channel. The data received by the MODEM would be passed through the serial interface to the protocol convertor and stored until the entire frame was received and converted. The output data would then be fed back to the MODEM through the serial interface. This implementation would be much more conservative in the use of hardware, but the required storage of the converted data would defeat the concept of the protocol convertor as a fast filter.

C. SUMMARY

A top-level floor plan for VLSI implementation of the parameterized protocol convertor was presented, detailing signal routes and circuit locations. Next a detailed analysis of the predicted signal delay through the two major circuits, the detect PLA and the CRC Shift Register,

was presented. The analysis shows both circuits are capable of operating at sufficiently high clock frequencies to allow use of the single byte storage method of protocol conversion. Two possible system-level implementations were presented, one for a continuous traffic load, and one for periodic message traffic.

VI. CONCLUSIONS

There is a need for a flexible, yet fast protocol convertor. In many instances, protocol standards have been misinterpreted, equipment has been forced into service it was not designed for, or systems have been developed without due consideration for interoperability. Any situation where two stations of a communication system can not communicate because of different protocols, no matter what the source of the difference in protocols, requires a protocol convertor.

The conflict between requirements for speed and flexibility in a protocol convertor can be resolved with a careful analysis of how much speed and what degree of flexibility is required. All systems have an inherent limitation in their communication speed capability, from one of many possible sources. If a protocol convertor can be designed to operate faster than the slowest component of the communication system it is designed to support, it can avoid becoming the bottleneck in the system. Dynamic flexibility can be sacrificed to increase this speed without limiting the application variations of the design if a limited degree of static flexibility is maintained.

The choice of which flow architecture to use also effects the speed of operation. If the design is patterned around a control flow architecture, the serial operation of the design will limit its possible speed. If the parallel constructs of a data flow architecture are used, the speed of operation can be greatly enhanced. The functions of a protocol convertor lend themselves to a parallel architecture approach, where many operations can take place simultaneously.

The design presented to meet the requirements of speed and flexibility has some, as of yet, unresolved problems. The major one being how to accomplish the bit and character stripping and stuffing. Early on in the analysis of the problem it was decided that to facilitate a high rate of data moving through the protocol convertor, there should be a minimum of storage of the data. Optimally, the minimum storage should be one bit, but the relative position of each bit in an eight bit byte is instrumental in determining the meaning of the bits. Because all eight bits are needed, the minimum storage is one byte. This minimum of storage is the bane of any stripping and stuffing circuit. The functions of stripping and stuffing are relatively straight forward when an entire frame is available in a storage buffer, but when only one byte at a time is available for manipulation, stripping and stuffing bits or characters becomes difficult.

As data is clocked into the protocol convertor, if a bit is stripped by the transparency circuit there is nothing to clock out of the protocol convertor when the empty bit interval arrives at the output. The concept of minimal storage requires data to be clocked out at the same rate it is clocked in. Only a single byte is ever stored within the convertor. If an entire character is to be stripped from the data, as in character oriented protocols, there would be a eight bit intervals without any data contents. To compensate for the empty bit intervals would require speeding up the input data or slowing down the output data, or both.

A similar problem occurs when bits or characters must be stuffed into the data by the transparency circuit. The data does not contain holes where these characters or bits are supposed to be inserted. Making room for the stuffed bits would require slowing down the input data or speeding up the data being output, or both. Serial, synchronous

transmissions are by design strictly clocked signals. The data is modulated with its own clock signal to aid in determining the segregation of the bits. The clock signal is constant, regardless of the requirements of a protocol convertor between the receiving station and the transmitting station.

Unfortunately, the transparency sub-function and consequently an ability to perform bit and character stripping and stuffing is required to implement a parameterized protocol convertor. The passage of transparent data between two stations is required in most applications. Ignoring the designation of transparent data as such would have far reaching effects on other aspects of the conversion process. For example, the character DLE used by character oriented protocols to indicate transparent data within the information field of a frame is not included in the block check characters. If the DLE character is not stripped before reaching the error control circuitry, the block check characters will never indicate an error free reception of the frame.

Another problem caused by the self-imposed requirement for minimal storage of the data occurs in the manipulation of byte count protocols. The length of the information field in a byte count protocol is included as part of the header contents. The header of a frame is transmitted first and subsequently arrives first at its destination. The interpretation of the byte count frame by the protocol convertor is no problem. The byte count is read from the header and a counter is set with the value. As the frame is clocked through the convertor, the passage of each byte decrements the counter by one. When the count reaches zero, the frame has been received in its entirety.

The transmission of byte count protocol frames is a different matter. The byte count of any protocol frame can only be determined by counting the bytes as they pass

through the convertor. The entire frame must be received before a byte count can be determined. The concept of minimal storage requires the converted data to be output as the input data is received. Only a one byte delay is allowed. The location of the byte count in the header of the frame, which is transmitted first requires a knowledge of the total number of bytes to be received while those bytes are still being input.

One possible solution to this problem would be to use a set frame length for output from the byte count protocol conversion units. For example, in a byte count frame, the maximum frame length is 16363 bytes of message data, ten bytes of frame header and two bytes of trailing block check characters. If the maximum frame length was assumed, the set byte count could be inserted in the frame header as it was output. Any frame space not utilized by the incoming data could be padded with blanks, zeros are some other predetermined character. The use of a set frame length, especially the maximum frame length allowable, would entail wasting a lot of frame space with shorter messages. The question remains if the wasted frame space is worth the completeness of design by allowing any input or output protocol framing technique.

Another problem is in the number and meaning of the control codes of the different protocols. To establish control code translation tables, a distinct relation must be established between the each control code of the two protocols involved. Some control codes have multiple meanings in a different protocol. For example, Receive Ready in a bit oriented protocol can be translated to Start Acknowledge or Positive Acknowledge in a byte count protocol. Bit oriented protocols respond with Receive Ready when they receive an initialization message and when the last frame was received without errors. The byte count protocols expect a Start Acknowledge control code in

response to any initialization message and a Positive Acknowledge control code in response to a correctly received frame.

A method must be devised to aid the protocol convertor in determining the intended meaning of the control code. The basis for one solution could be narrowing down the possible responses to each control code to be detected by the protocol convertor. For example, in the byte count protocol conversion unit, once the Start Message control code is sent, a Start Acknowledgment control code is expected in response. A bit oriented protocol sends the same Receive Ready control code in response to both a Start Message control code and an error free frame. The byte count protocol conversion unit that initiated the Start Message knows what to expect as a response. The conversion unit should interpret the Receive Ready control code as a Start Acknowledgment control code, which makes sense, instead of Positive Acknowledgment, which does not.

There is still much work to be done in developing an optimum protocol convertor circuit. The myriad of problems presented by the lack of corresponding functions between the different protocols presents a challenge that will exist for a long time. Until a single standard can be developed, clearly delineated, and accepted, there will be a need for protocol conversion.

LIST OF REFERENCES

1. Stallings, W., Computer Communications: Architectures, Protocols, and Standards, IEEE Computer Society Press, 1985.

2. Tanenbaum, A. S., Computer Networks, Prentice-Hall, Inc., 1981.

3. McNamara, J. E., Technical Aspects of Data Communication, Digital Press, 1982.

4. Tanenbaum, A. S., "Network Protocols", Computing Surveys, Vol. 13, No. 4, Dec. 1981.

5. Karten, H. A., "Surviving the Stampede", Systems User, pp.7-10, June 1984.

6. Joint Chiefs of Staff Publication 1, Dictionary of United States Military Terms for Joint Usage, 1 June 1979.

7. Gabel, D., "Communication Software: Quicker Data-Transfer Speeds and Complex Error-Checking Protocols Make the Choices More Difficult," PC Week, September 24, pp. 59-61, 1985.

8. Robinson, P., "Protocol Converters: the Answer to Compatibility Problems," Computer Communications, Vol. 5, No. 3, June 1982.

9. Paseman, W. G., "Applying Data Flow in the Real World," Byte, pp.201-214, May 1985.

10. Bracker Jr., W. E., "Surveying the Protocol Conversion Vendor's Offerings," Data Communications, Vol.12, No.8, pp. 89-101, August 1983.

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

12. Senior, J. M., Optical Fiber Communications, Prentice/Hall, 1985.

13. Johsi, S. and Iyer, V., "Protocols and Chips: Can There be a Synergism," Electro '83 Conference Record, Sec. 10/2, pp.1-8, 1983

14. Hwang, K., Supercomputers: Design and Applications, IEEE Computer Society, August 1984.

15. Hwang, K. and Briggs, F. A., Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

16. Short, K. L., Microprocessors and Programmed Logic, Prentice-Hall, 1981.

17. Sarnak, N. and Jaffe, E., "8087 Performance Considerations," PC Tech Journal, pp. 30-47, Sept/Oct 1983.

18. Gajski, D. D. and others, "A Second Opinion on Data Flow Machines and Languages," Computer, pp. 58-70, Feb. 1982.

19. Stone, H. S., Microcomputer Interfacing, Addison-Wesley, 1983.

20. Mead, C. and Conway, L., Introduction to VLSI Design, Addison-Wesley, 1980.

21. Newkirk, J. and Mathews, R., The VLSI Designer's Library, Addison-Wesley, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Prof. M. L. Cotton Code 62Cc Naval Postgraduate School Monterey, California 93943	2
4.	Department of the Navy Headquarters, U. S. Marine Corps (LMC-1) Attn: Capt. B. Garris Washington, D.C. 20380-0001	3
5.	Department of Electrical & Computer Engineering Code 62 Naval Postgraduate School Monterey, California 93943	1

216457

Thesis
G216
c.1

Garris
Analysis and design
of a parameterized
protocol convertor.