AN ABSTRACT OF THE THESIS OF

Shashank C. Merchant for the degree of Master of Science in

Electrical and Computer Engineering presented on June 4, 1992.

Title: A Programmable Network Interface Unit for Hybrid Meshnet Local Area Networks

Redacted for Privacy

Abstract approved: _____
James H. Herzog

A Hybrid Meshnet LAN, a new local area network architecture, has been proposed by Dr. Cheoul-Shin Kang and Dr. James Herzog. It provides for distributed control hybrid architecture which is good for effective load sharing under various local area network environments. Hybrid Meshnet has a dual channel structure, a token ring and a collection of full-duplex data links. The multiple high-speed transmissions, private and secure communications and large volume of data transmission capability are some of the features of Hybrid Meshnet.

The design features necessary to implement the network interface unit (NIU) for the Hybrid Meshnet are presented. The unit is a multiprocessor system with each processor having a RISC-like architecture. Various asynchronous activities are distributed among the three processors resulting in a balanced network interface unit.

Except for the time critical and non-varying functions, all the functionalities of the unit are programmable. The hybrid meshnet protocol is still in the development stage. The programmable unit will accommodate the changes in the protocol. The network interface unit will be compatible with most of the host computer systems.

The study is one step forward in the direction of Hybrid Meshnet Local Area Network's implementation.

A Programmable Network Interface Unit For Hybrid Meshnet Local Area Networks

by

Shashank C. Merchant

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed June 4, 1992

Commencement June 1993

APPROVED:

Redacted for Privacy

Associate Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented June 4, 1992.

Typed by Shashank C. Merchant.

ACKNOWLEDGEMENTS

I would like to express my appreciation to Prof. James Herzog for his help, patience, advice and encouragement. I am grateful to Prof. Bella Bose, Prof. Shih-Lien Lu and Prof. Dwight Bushnell for agreeing to be on my graduate committee and taking time out of their busy schedules to be present at my defense.

My special thanks and gratitude to Prof. Otto Gygax for providing me with an opportunity to gain invaluable experience as a system administrator in his team and above all supporting me financially through my education at OSU.

I, also thank Prof. Lenders for his initial support and continued good wishes.

I thank the office staff of the Electrical and Computer engineering department and specially Rita Wells for their full-hearted cooperation during my stay here.

I would like to thank Ronak Shah, Raj Shroff, Lalit Merani and Manoj Raisinghani for their continued support and friendship.

Finally, and above all, I am grateful to thank my parents, brother and sister-in-law without whose encouragement and blessings this would surely not have been possible.

TABLE OF CONTENTS

1. INTRODUCTION 1
   1.1 Local Area Networks 1
      1.1.1 Definition 1
      1.1.2 Network Control 2
      1.1.3 Message Transport Technologies 3
      1.1.4 Network Structure and Topology 3
         1.1.4.1 Star Topology 4
         1.1.4.2 Bus Topology 5
         1.1.4.3 Ring Topology 5
   1.2 Hybrid Meshnet: A Local Area Network with Distributed Control 6
   1.3 Scope of the study 7
   1.4 Organization 9
2. LITERATURE SURVEY 11
   2.1 Communication Protocols 11
   2.2 Background 14
      2.2.1 Token Ring 14
      2.2.2 Mesh Network 15
      2.2.3 Flood Routing 15
   2.3 Hybrid Meshnet 16
      2.3.1 Network Connection 16
      2.3.2 Network Control and Routing Mechanisms 17
         2.3.2.1 Broadcast Type Data Transmission 18
         2.3.2.2 Shortest Path Flood Routing 18
         2.3.2.3 Learning Mode Flood Routing Technique 20
         2.3.2.4 Fault Routing Using the Routing Table 20
3. DESIGN ISSUES 22
   3.1 Architectural View 22
   3.2 Design of a Network Interface Unit 23
   3.3 Organization of the Network Interface Unit 25
   3.4 Modifications 28
4. DESIGN FEATURES OF THE NIU 32
   4.1 Organization of the Network Interface Unit 32
   4.2 Design Strategies for the Processors 39
   4.3 Data Processor 46
   4.4 Token Ring Interface 52
   4.5 Data Link Interface 62
5. PERFORMANCE EVALUATION 74
   5.1 Results 76
   5.2 Determination of BDRN, THT and TNT 81
   5.3 Processor Utilization 83
6. RESULTS AND CONCLUSIONS 84
   6.1 Areas of Further Study 85

BIBLIOGRAPHY 88

APPENDIX 90

LIST OF FIGURES

Figure		Page
1.1	Star Topology	4
1.2	Bus Topology	4
1.3	Ring Topology	6
1.4	A Possible Network Configuration of Hybrid Meshnet	7
2.1	Open Systems Interconnection (OSI) Reference Architecture	12
2.2	Relationship between OSI/RM and LAN/RM	14
2.3	Mesh Topology	15
2.4	Concurrent Communication in Hybrid Meshnet	18
3.1	Functional Diagram of the Network Interface Unit	24
3.2	Network Interface Unit from the Network's View	24
3.3a	Shared Bus Configuration (Single Processor NIU)	27
3.3b	Shared Bus Configuration (Multi-Processor NIU)	27
3.3c	Multi-Processor NIU with Adapter Bus	27
3.4	Token Ring Data/Control Frame	30
3.5	CACK Frames	30
4.1	The Network Interface Unit	33
4.2	Host System Interface for DMA	34
4.3	Shared Memory Organization	35
4.4	Relationship between LAN/RM Model and the NIU	37
4.5	General Flow of Control and Data in NIU for Broadcast Transmission and Flood Routing	38
4.6	Simplified Block Diagram of the Basic Architecture	40
4.7	Basic Architecture of the Processors	41
4.8	Events on Every Stage of the Pipeline	43
4.9	Resource Allocation for Different Instruction Types	45
4.10	Working of the Data Processor	47
4.11	Programmable Registers for the Data Processor	49
4.12a	Decision Table (256x32)	49
4.12b	Routing Table (256x32)	49
4.13	Instruction Format for the Data Processor	51
4.14	Instruction Set for the Data Processor	51
4.15	Logical Operation of the Token Ring Interface	53

LIST OF FIGURES
(continued)

Figure		Page
4.16	Basic Block Diagram of the Token Ring Interface	53
4.17	Working of the Token Ring Interface	54
4.18	Programmable Registers for the Token Ring Interface	56
4.19	Instruction Format for the Token Ring Interface	58
4.20	Instruction Set for the Token Ring Interface	58
4.21	Receive and Transmit Sections of the Token Ring Interface	59
4.22	Block Diagram of the Data Link Interface	62
4.23	Working of the Data Link Interface	63
4.24	Instruction Format for the Data Link Interface	64
4.25	Instruction Set for the Data Link Interface	65
4.26	Programmable Registers for the Data Link Interface	67
4.27a	Transmit Section of the Data Link Interface	69
4.27b	Receive Section of the Data Link Interface	69
4.28	Switching Block for the Data Link Interface	70
4.29	Switching Section of the Data Link Interface	72
5.1	Message Transfer Time for Broadcast Transmission	77
5.2	Setup time in Flood Routing	78
5.3	Setup Time and Message Transfer Time for Flood Routing	78
5.4	Comparison between Broadcast Transmission and Flood Routing	80
5.5	Comparison between Broadcast Transmission and Flood Routing	80
5.6	Message Transfer Time for Learning Mode Routing	82
5.7	Message Transfer Time for Fault Routing	82

A PROGRAMMABLE NETWORK INTERFACE UNIT
FOR
HYBRID MESHNET LOCAL AREA NETWORKS

CHAPTER 1
INTRODUCTION

1.1 Local Area Networks

1.1.1 Definition

A Local Area Network is a data communication system which allows a number of independent devices to communicate with each other. A Local Area Network (LAN) is distinguished from other types of data networks in that the communication is usually confined to a moderate size geographic area such as a single office building, an industrial complex or a campus. A classification based on distance may be used to describe the notational boundaries as follows:

- 0-10m Computer Peripherals.
- 10m-10km Local Area Networks.
- 10km+ Wide Area Networks.

LANs occupy the middle ground between tightly coupled components of individual computer systems and the traditional loosely coupled communication networks already widely in use.

LANs generally have three distinctive characteristics [4] :

- A diameter of not more than a few kilometers.
- A total data rate exceeding 1Mbps.

- Ownership by a single organization.

LANs have figured prominently in research and development for more than ten years and commercial exploitation has been increasingly active. They are used to provide relatively inexpensive communication between workstations and devices, distributed processing, rapid access to distributed data banks and sharing of expensive devices and resources.

1.1.2 Network Control

A node is a physical device that may be attached to a shared medium local area network for the purpose of transmitting and receiving information on that shared medium. The network control mechanism has the responsibility for all activities associated with network communication. When network resources are shared by multiple nodes there must be some method whereby a particular unit requests a resource and is allowed an access to the resource at a given time. The control scheme for managing the resources can be centralized or distributed.

In a *centralized control* system all network activities are governed by a central controller. In this approach processing overhead at each node can be greatly reduced, but control operations become vulnerable to central node failure.

In a *distributed control* system, a distributed algorithm running at each node controls the whole network by interacting and cooperating among themselves. The design of the distributed algorithm is relatively difficult. The distributed approach provides a greater reliability in terms of computer failure at the expense of more processing at each node.

1.1.3 Message Transport Technologies

Networks may be conveniently classified by the technique employed in transporting messages between the nodes. Let us look at each one of them.

A *circuit-switched* network transmits a message by setting up a dedicated path between the source and destination. A special control message sent from the originating node to the destination node sets up the path. The entire fixed-delay path is allocated to a particular transmission until the path is released. No processing of messages is required at intermediate nodes and hence there are no store-and-forward delays. Circuit-switched networks provide a fast, efficient message transmission once the circuit has been allocated. It is a particularly attractive mode for large-volume data transmissions.

A *message-switched* network involves message transmission by moving it through various transmission links and message buffers. A message is stored and then transmitted to the next node along the message path. The path may be fixed or determined dynamically as the message progresses towards its destination node.

A *packet-switched* network differs from a message-switched network in that long messages are first decomposed into fixed-size segments called packets. These packets independently traverse the network until they reach the desired node, where they are reassembled into corresponding message.

1.1.4 Network Structure and Topology

The nodes of a computer communication network can be connected in a variety of ways. The term topology, refers to the way in which the nodes, transmission links and user devices are organized in a communication network. The topology of the interconnections in a network is of primary concern. The popularity and use of a particular topology depends on many factors [1]: performance, cost, reliability,

flexibility, modularity and reconfigurability of the network, as well as network control strategies. Among various topologies the following have been widely used.

1.1.4.1 Star Topology

A star network consists of nodes connected to a central switch via a single bi-directional link (Figure 1.1). The messages between any two nodes has to pass through the central switch. A star network has the following characteristics :

- Network operation is highly dependent upon the correct operation of the central node. Central node failure results in failure of the entire network.
- A dedicated path is established between the source and destination.
- A single node failure doesn't affect the network performance or operation.
- It is easy to add extra nodes. The capacity of the central switch is the only limiting factor.

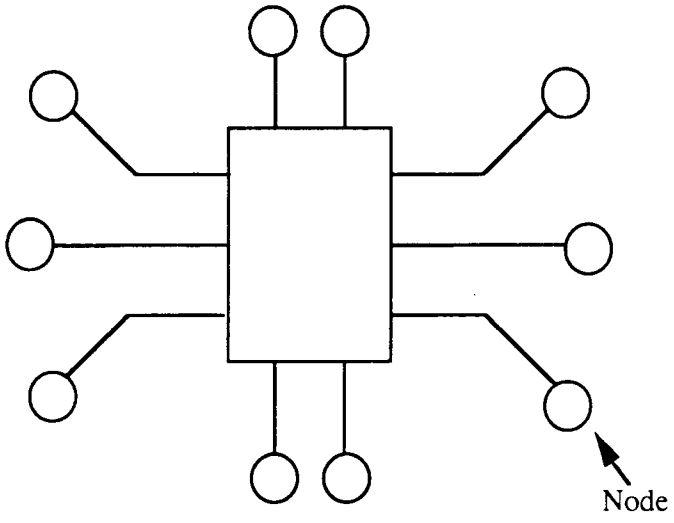


Figure 1.1 Star Topology

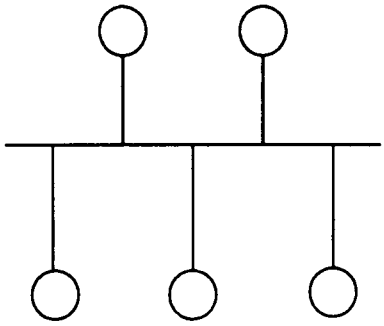


Figure 1.2 Bus Topology

1.1.4.2 Bus Topology

A bus network consists of a single transmission path for communication among the nodes (Figure 1.2). Some method must be provided to control the access of the media by the nodes. The most common techniques for this topology are Carrier Sense Multiple Access (CSMA) and token bus. A bus topology network has the following characteristics:

- A single transmission medium and inherent broadcast transmission capability.
- Requirement of arbitration to access the medium. The arbitration scheme may be complex.
- Node failure does not affect the network. However single faulty node can reduce the network performance drastically.
- Suitability for high-speed burst traffic.

1.1.4.3 Ring Topology

The basic organization of a ring is simple. Several nodes are linked together to form the equivalent of a ring (Figure 1.3). It provides for high channel utilization and reduces the burden of routing or arbitration. Token ring is a typical example of systems which utilize ring topology. A ring topology usually has the following characteristics :

- Distributed network control.
- The message is circulated through all nodes.
- Inherent broadcast transmission capability.
- Provides simple access control for usage of transmission link.
- The network is susceptible to a single link/node failure and a lost token.
- Requires to bring down the network for addition of new nodes.

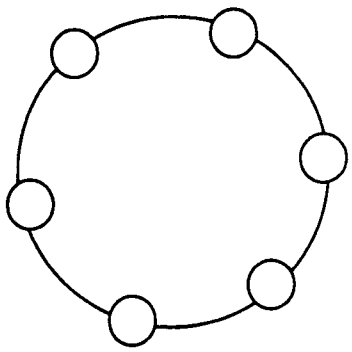


Figure 1.3 Ring Topology

1.2 Hybrid Meshnet: A Local Area Network with Distributed Control.

The above presented topologies have many advantages :

- Relatively simple network control.
- No formal routing mechanism required.
- Single transmission medium.
- Structured topology.

Both bus and ring implementations suffer from some serious disadvantages :

- Limitations on performance characteristics due to a lack of concurrent communications.
- Lack of guaranteed private and secure communications.
- Performance degradation under increased network traffic.
- Limitation on message/packet length.

Recently C.S. Kang and James H.Herzog [2] suggested a new distributed control hybrid architecture local area network, Hybrid Meshnet, which is good for effective load sharing under various LAN application environments (Figure 1.4). Hybrid Meshnet has a dual channel structure, a token ring and a collection of full-duplex data links.

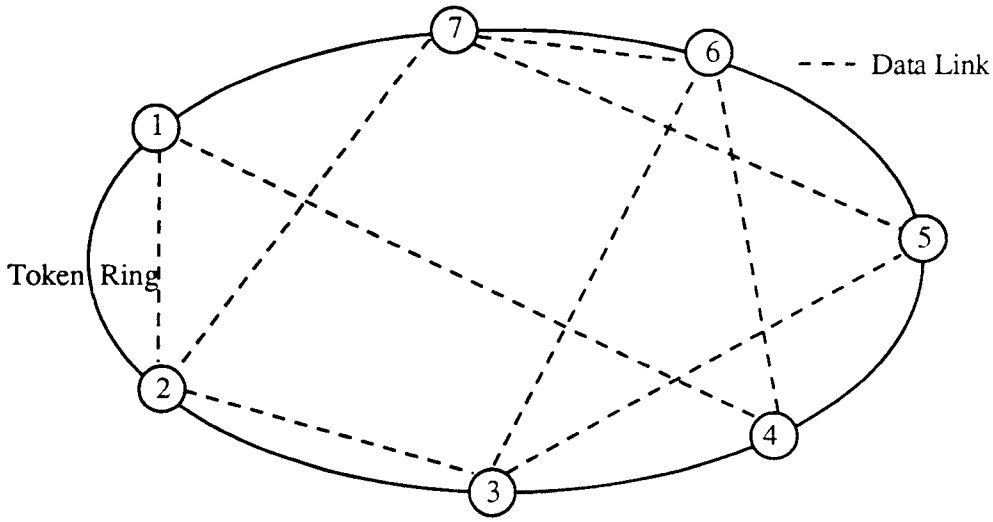


Figure 1.4 A Possible Network Configuration of Hybrid Meshnet.

Many applications require prompt, robust and flexible services and their environments are subject to constant change and expansion. For instance the number of nodes may be increased but the response time must remain at an acceptable level. Moreover certain environments may also be changed through breakdown, repair, or user reconfiguration. Hybrid Meshnet is designed to match these requirements [1].

Hybrid Meshnet is well suited for small to moderate sized shared resource/distribution processing and control systems put together within a single building or complex of buildings for the purpose of data processing, office automation, factory automation and various kinds of process controls. These systems may be characterized by brief messages, such as control signals, process control commands, or alarms, and large volumes, such as exist in file transfer systems [1].

1.3 Scope of the Study

Physically, a network consists of a communication channel to which a number of intelligent nodes are coupled. The intelligent nodes have several functions; they must transmit and receive on the channels; they must control access to the channels; and they

must interface the devices coupled to the network at the node [10] . Three types of design problems arise :

1. Design of network architecture and protocols.
2. Design of the nodal hardware; and
3. Design of the nodal software.

The first of the problems has been addressed and well handled in [1] for Hybrid Meshnet. The architecture and protocols proposed are taken as the basis for this study. However, some liberty has been taken to modify the protocols where situation warranted. The main thrust of this study is to approach the other two problems.

Hybrid Meshnet is only one of the various hybrid architectures possible. The problems are tackled in a manner that the study carried out for Hybrid Meshnet is applicable to other hybrid architectures.

Simulation results done in [1] indicate that the Hybrid Meshnet can be an effective LAN architecture. Purpose of this study is to move one step forward. An effort will be made to give the architectural specifications for the network interface unit (NIU). Various design issues for the NIU will be considered. Suggestions regarding the overall configuration of the NIU, the processor architecture, their instruction sets and the working will be made. The study was carried out with following in mind.

- Programmability : Making the unit programmable will provide flexibility in changing or adding new functionalities to the present system. This will also result in simpler control units.
- Modularity and Expandability : The interface unit is designed to allow compatibility with most of the existing computer systems with the host computer having no knowledge of the underlying network. The unit should be designed such that it is not a limiting factor in performance.

- In this work, a network interface unit will be designed to perform processing actions associated with the hybrid meshnet. Optimization of the design will be reserved for future work. The NIU will form the media access control (MAC) and logical link control layers (LLC) of the LAN model. This layers will be explained in Chapter II.
- Approximate performance measurements will be performed based on instruction timings and hardware delays. From this the speed of operation will be determined.

There are various modes of routing in hybrid meshnet (refer to Chapter 2). The performance and timing requirements (setup time, propagation delay and transmission time) for each of the routing mechanism will be estimated. The exercise will also enable us to suggest optimum values for various programmable registers. Analysis of sample routines written for the interfaces of the NIU will help us to achieve these objectives.

The organization of the NIU, the processor architectures, the timing and performance measurements carried out as a part of the study will bring us one step closer to the actual implementation. Along with [1], this will form the basis for further study. Overall, the process will also allow us to determine the feasibility, complexity and cost involved in implementing hybrid architectures, specially Hybrid Meshnet, for LANs.

1.4 Organization

The first chapter has served to present the basic concepts of local area networks. Hybrid Meshnet has been introduced and scope of the study defined. The following chapter will discuss the communication protocols and layering scheme. Hybrid Meshnet will be discussed in detail. Chapter 3 will provide the insight into the various design issues involved. Chapter 4 discusses the implementation of the Network Interface Unit (NIU) for Hybrid Meshnet. Chapter 5 evaluates the design and steps taken in the study.

The final chapter presents the conclusions and identifies various possibilities open to further the research.

CHAPTER 2
LITERATURE SURVEY

2.1 Communication Protocols

A network consists of a collection of interconnected nodes that permit the exchange of units of information or data, among each other. An orderly exchange of data requires that each node conform to some pre-established agreements or rules. These rules specify the formats and relative timings of messages to be exchanged among the nodes. A network protocol establishes these rules, standards, or conventions. It essentially consist of three elements : (1) syntax, or the structure of data and control messages; (2) semantics, or the set of control messages to be issued, actions to be performed, and responses to be returned; and (3) timing, or specification of the order of event executions [12].

Handling communication between various applications running on different computers in an heterogeneous environment is a complex task. The problem is required to be broken into manageable parts instead of being handled as a single unit. The concept of layers accommodates this idea. The underlying principle is that a given layer in a node logically exchanges messages with its corresponding layer in another node, and that processing at the other layer is transparent to it. There are peer-to-peer protocols for communication between the corresponding layers in different nodes. Also, each layer relies on the next lower layer to perform more primitive functions and provide services to the next layer. This concept of layering yields various advantages [12] :

1. It allows interaction between functionally paired layers in different nodes.
2. It leads the protocol software to simpler descriptive, development and testing processes.

3. It permits changes or modifications to a given layer without affecting other layers.
4. It provides simpler management of, and smoother enhancements to, the protocols.

The most accepted layering scheme is the Open Systems Interconnection (OSI) reference model architecture shown in Figure 2.1.

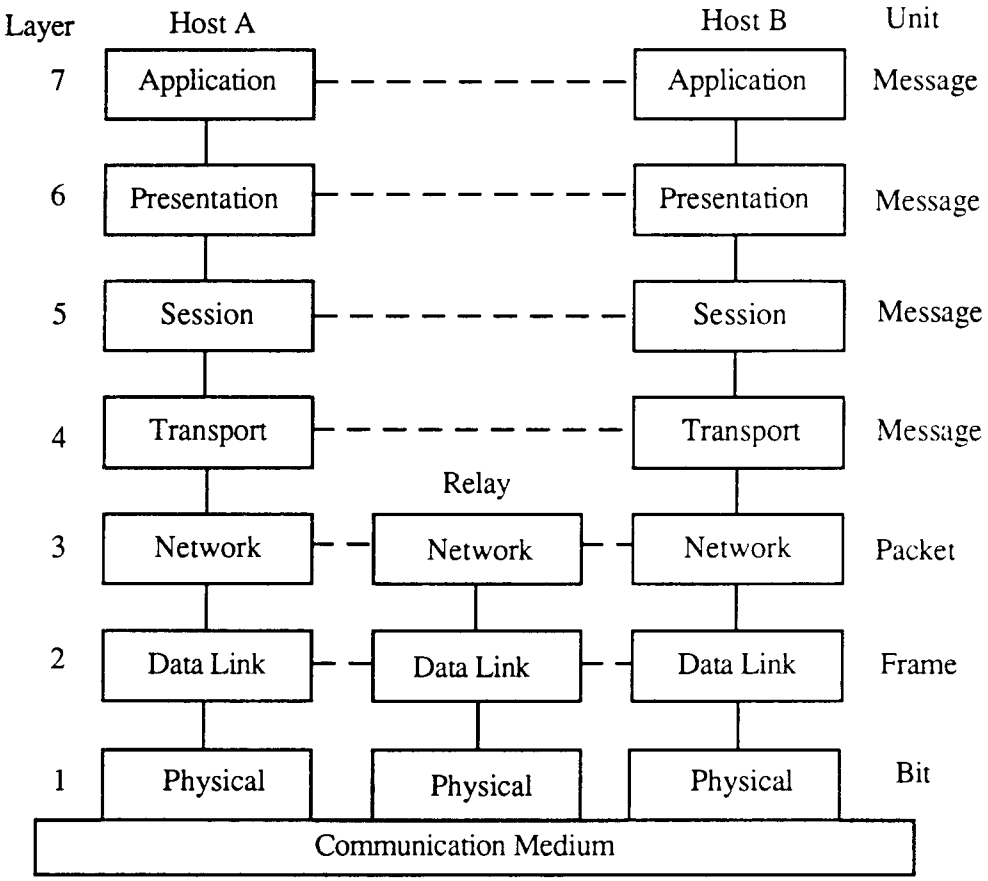


Figure 2.1 Open Systems Interconnection (OSI) Reference Architecture [13].

Figure 2.1 shows a classical representation of the communication between two systems at A and B connected by a relay. Each end system is defined by seven layers as shown, with well defined interfaces between the layers. The seven layers can be summarized as follows: [15]

1. Physical Layer : This layer is concerned with the electrical and mechanical means of transmitting and receiving information using a particular communication medium.
2. Data Link Layer: This layer is responsible for maintaining the integrity of information sent between two points. It provides for an error detection and correction scheme.
3. Network Layer : This layer has the responsibility of ensuring that information is transferred correctly over the network. It involves setting up and maintaining the necessary links. It provides routing functions for transfer of information from a source to one or more destinations.
4. Transport Layer : This layer provides the reliable, transparent transfer of data between end points.
5. Session Layer : This layer is responsible for establishing, managing and terminating a connection (session) between two processes.
6. Presentation Layer : This layer performs generally useful transformations on data to provide a standardized application interface and to provide common communication services.
7. Application Layer : This layer provides services to the user of the OSI environment. Corresponding to the OSI reference model (OSI/RM) is the local area network reference model (LAN/RM).

Figure 2.2 shows the relationship between the two reference models. Basically the LAN/RM has only three layers: these correspond to the lowermost two layers of the OSI/RM [4] [11] :

1. Physical Layer : This is similar to the physical layer in OSI/RM.
2. Media Access Layer (MAC) : This layer provides a means of sharing a common medium among a number of different devices. Such control is necessary to allow

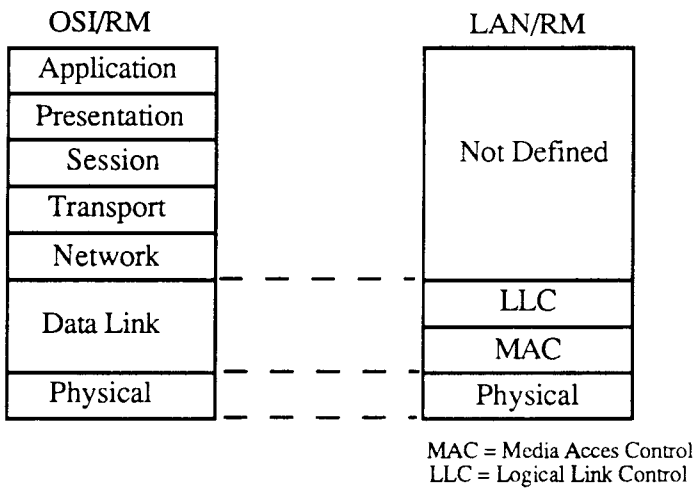


Figure 2.2 Relationship between OSI/RM and LAN/RM [4].

for the efficient utilization of available bandwidth and to avoid deadlock when competing for a single bandwidth.

3. Logical Link Control Layer (LLC) : This layer is concerned with establishing, maintaining and terminating a logical link between devices. It provides services for both connection oriented and datagram type of communications. The datagram type involves messages being sent on a best effort basis without any acknowledgement.

2.2 Background

2.2.1 Token Ring

Token ring is the medium access control protocol standard (IEEE 802.5) for ring topology as specified by the IEEE 802 standard committee. In token ring, a single token frame (consisting of unique control bits) circulates around the ring in the absence of any message. A node wishing to transmit must wait until it detects a token passing by. It then changes the token from "free token" to "busy token" and appends the message. This is the token hold state.

During the token hold state, no free token frame is circulating on the ring. A station wising to transmit must wait. The message passes from node to node. At the destination the message is copied for local use and also forwarded. The message loops back to the source and is removed from the ring at this point. The source node will then insert a new free token frame on the ring.

The use of a token guarantees that only one node can transmit at a time. When a transmitting node releases the token, the next node downstream with data to send will be able to grab the token and transmit.

2.2.2 Mesh Network

A mesh network is shown in Figure 2.3. A Mesh topology provides capabilities to handle exceptional communication loads and a high degree of fault tolerance by providing alternate paths between the nodes. Mesh topology networks may result in better performance for specific applications than do networks based on simple bus or ring topology.

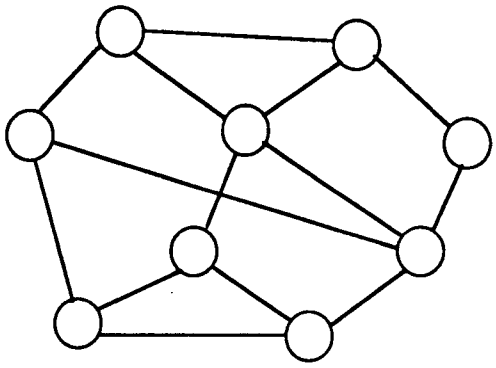


Figure 2.3 Mesh Topology

2.2.3 Flood Routing

Routing defines the mode of passing data from one node to the other in order that the packet transmitted from the source reaches the destination. Flood routing is the

routing method in which the packets are flooded over the network [18]. At each node, the received packet is checked for duplication. If the packet was already received, it is discarded; else it is forwarded to all its neighbors except the one from which it was received. This is repeated at each node until it reaches the destination node. This leads to a packet arriving at the destination via the shortest available route. The removal of duplicate packets ensures that they remain on the network only for a finite time. This scheme is capable of finding alternate routes in event of a node/link failure along the shortest route and thus lending itself to fault tolerance. Some of the overheads involved are due to the checking and duplication of packets.

2.3 Hybrid Meshnet

2.3.1 Network Connection

Each node of Hybrid Meshnet is connected to two communication channels: a token ring and data channel (data links). The primary purpose of the token ring is to exchange control messages and short data messages among the network nodes. Most of the network control activities are handled by exchanging proper sequences of control messages through the token ring. The data channel is a collection of point-to-point data links interconnecting pairs of nodes, the primary use of which is to exchange long data messages between network nodes [1].

The token ring, is a modification of IEEE LAN 802.5 standard [3]. For broadcast-type data transmission, it implements a standard token-ring protocol. Transmission of messages over the data links require some changes in the protocol. In this case, the token ring serves to pass a series of control messages to the nodes in the network. The modified protocol allows the destination to send a control message over the ring without grabbing a free token.

The data channel is a point-to-point bi-directional link connecting a pair of nodes in the network. The organization of the data channels depends on the data communication requirement. As per the needs, links can be added or removed. Any input link can be dynamically linked to one or more output links through a switching device in the node. At any given time, there can be more than one path between any two nodes. The routing mechanism (flood routing) for the data links has the ability to dynamically establish the shortest path among the available paths between source and destination.

2.3.2 Network Control and Routing Mechanisms

Hybrid meshnet employs a distributed control mechanism. This means that any node on the network can assume control of the network. This is done by grabbing the control token circulating on the token ring which will allow the node to initiate a transmission. The message can be transmitted on either the ring or through the data channels. As in token ring, only one node can initiate a transmission at any given time. However, unlike token ring, multiple transmissions are possible. Suppose node 3 requires to transmit message to node 6 (refer to Figure 1.4). A data link path between the source and destination is established by the exchange of control messages over the ring. Once the path is setup, the data message transmission occurs on the data link. Token ring is now free. If node 1 desires to transfer a message to node 4, it can follow the same procedure. Multiple data message transmissions are now active on the network (Figure 2.4). The upper limit on the number of possible concurrent transmissions is set by the number of data links, the number of nodes and the message lengths.

There are three communication methods used in the hybrid meshnet routing mechanism : broadcast data transmission, a flood routing and a fault routing using a routing table. The decision on the type of routing selected for a given message is dynamic

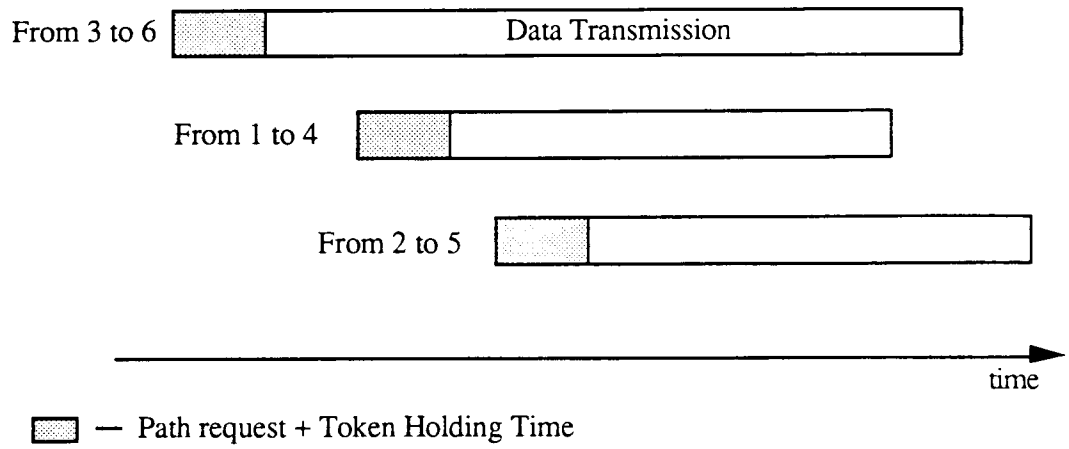


Figure 2.4 Concurrent Communication in Hybrid Meshnet.

and depends on the length, priority and address of the destination node among various other factors. This will be explained in detail while considering the design issues in later chapters. Let us consider each one of the routing mechanism in detail.

2.3.2.1 Broadcast Type Data Transmission

Broadcast data transmission is used to directly transmit data messages via the token ring. No routing is necessary and this method does not involve any switching or path setup times. In hybrid meshnet, broadcast transmission serves to send control messages over the ring. Short data messages can also be transmitted using this method. However, long message transmissions on the token ring reduces the efficient channel allocation among the nodes. The upper limit on the message length is also set by an implementation. Broadcast transmission is the only choice when no data link path exists between the source and destination.

2.3.2.2 Shortest Path Flood Routing

Shortest path flood routing is the principal data transmission method for long data messages. The essential features of this control strategy include the following operation

steps.

(1) *Path Request* : A node, ready to transmit, grabs the token and sends a path control message over the ring. It indicates that the source is ready to transmit data messages over the data links. The control message also contains a frame indicating the destination.

(2) *Path Search and Data Message Transmission* : When the source receives the control message back after traversing the ring, it floods a short data message over all of its available free outgoing links. The flooding is continued at each node. There is no need for destination address comparison time, since the source and destination have already been announced to all nodes by the path request. If a route exists, the message will reach the destination via the shortest available path. A time out occurs if the message doesn't reach the destination within a prespecified time limit. This may be due to an absence of a path from source to destination or as a result of a busy path.

(3) *Path Set-up and Freeing Tentatively Reserved Components* : When the message is received by the destination node, the destination node sends a Connection Acknowledgement (CACK) to the outgoing link from which it received the message. This is then repeated at each node present in the shortest path found by the flooding technique. Each node receiving the CACK frees up all the reserved links except for the one which is acknowledged.

The destination node also broadcasts a Path Grant control message on the token ring. It allows all the nodes, not in the shortest path, to release the links reserved during the initiation of routing. It also indicates that the data path has been established. Each node updates its tables to indicate the busy source and destination.

(4) *Token Passing* : On reception of the CACK and Path Grant control message the source node releases the token and continues to transmit data over the data link. In absence of either path grant or CACK within the time out period, the source node stops transmitting and passes the token to the next node.

(5) *Data Message Acknowledgement and Path Release* : The destination node sends a Data Message Acknowledgement (DACK) backwards on a data link path on reception of the complete data message. This allows all the nodes in the path to release the links. The destination node also broadcasts an acknowledgement control message over the ring, on reception of a token.

2.3.2.3 Learning Mode Flood Routing Technique

The learning mode routing technique allows a node to learn more about the network in relation to itself. The basic steps for the routing are similar to flood routing. However as the path set-up message is passed from node to node, each node attaches its address to the frame. When the message reaches the destination, it will have the information about the shortest path to the source. The destination node sends a CACK on the setup link while simultaneously broadcasting a path grant control message over the token ring. The path grant control message provides the information learned during the setup to all the nodes. Because of extra overhead, this mode of routing is used only when there is no network load. The information learned is kept in the routing table at each node in the network.

2.3.2.4 Fault Routing Using the Routing Table

The Fault routing method for fault diagnosis uses a routing table. The routing table has the information gathered during a learning mode routing about a path from a source

to a destination. Based on the information, it provides the address of the nodes forming the path between the source and destination in the path request control message. Consequently only the involved nodes are switched to receive the message. If a node in the path is faulty, the message will not reach the destination. If there is no CACK within a length of time equivalent to a time-out period, a faulty unit within the data path is indicated. Using sophisticated algorithms, it is possible to determine this faulty unit. The fault routing does not involve transmission of any data messages and is used under the condition of no traffic over the network.

CHAPTER 3
DESIGN ISSUES

3.1 Architectural View

There are two important ways to view local area network design : architectural, which emphasizes the logical divisions of the system and how they fit together, and implementational, which emphasizes the actual components, and their interconnection [9]. Section 2.1 presents the LAN/RM model and describes the functions of each layer. The conventional local area network topologies like token ring and ethernet have characteristics which conforms to this model:

1. All the nodes in the network share a single physical medium or channel.
2. No routing or switching functions are necessary for information transfer from one node to the other.

It can be seen that a layer corresponding to the network layer (for routing and switching functions) of ISO/RM model is not required for these topologies. However in case of Hybrid Meshnet LAN :

1. Nodes may be connected to more than one physical media. Each node is a part of the token ring but can also be connected to other nodes through data links.
2. Nodes are required to make the decision on using the more suitable medium for the given message. In case of data links, the packets need to be routed while being transmitted from the source to the destination node.

The hybrid meshnet LAN requires more functionalities from the layers of LAN/RM model than the existing LANs. It is required to accommodate the routing and switching functions into the existing layering scheme. One of two approaches is possible.

- It can be assumed that there exists a layer above the LLC layer which will perform the necessary routing and decisions for the node. Since these functions are similar to that performed by the network layer of OSI/RM model, we can call this layer a network layer.
- The network routing and decision functions can be considered as extensions to the function set defined for the LLC layer in the LAN/RM model.

The second approach leads to a more uniform mapping to the proposed Network Interface Unit. The LLC layer will also provide functions for fault diagnosis. The network interface unit to be designed will implement the LLC and MAC layers. Some of the LLC functions are assumed to be carried out in software by the host. The mapping of the layers vis a vis the network interface unit will be presented in the next chapter.

3.2 Design of a Network Interface Unit

There are three parts to communication systems. The first incorporates the actual systems that communicate with each other and the second is the physical means to connect them - such as cables, microwave technology or fiber optics. The third area forms the interface between the systems and physical links. It is called the Network Interface Unit (NIU). The NIU of Hybrid Meshnet consists of three functional parts [1]:

1. *A data link interface* : This consists of a switching device, a data link interface transmitter, a data link interface receiver and a data link interface control.
2. *A token ring interface* : This is composed of a token ring interface transmitter, a token ring interface receiver and a token ring control.
3. *An attached component interface.*

These functional parts are shown in Figure 3.1 and their relationship with the entire network system is shown in Figure 3.2.

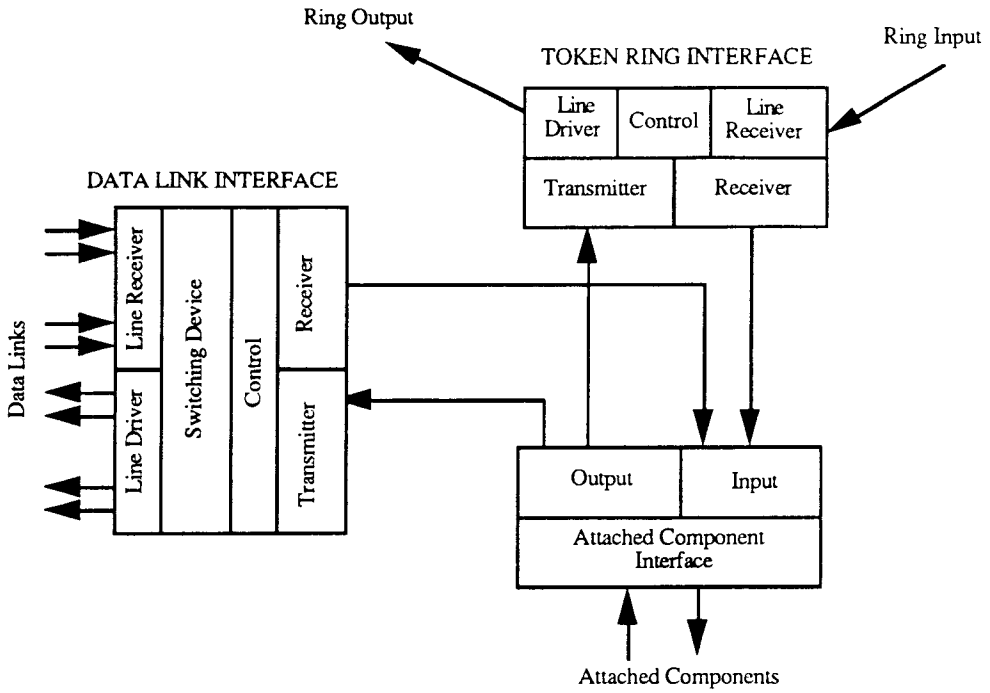


Figure 3.1 Functional Diagram of the Network Interface Unit

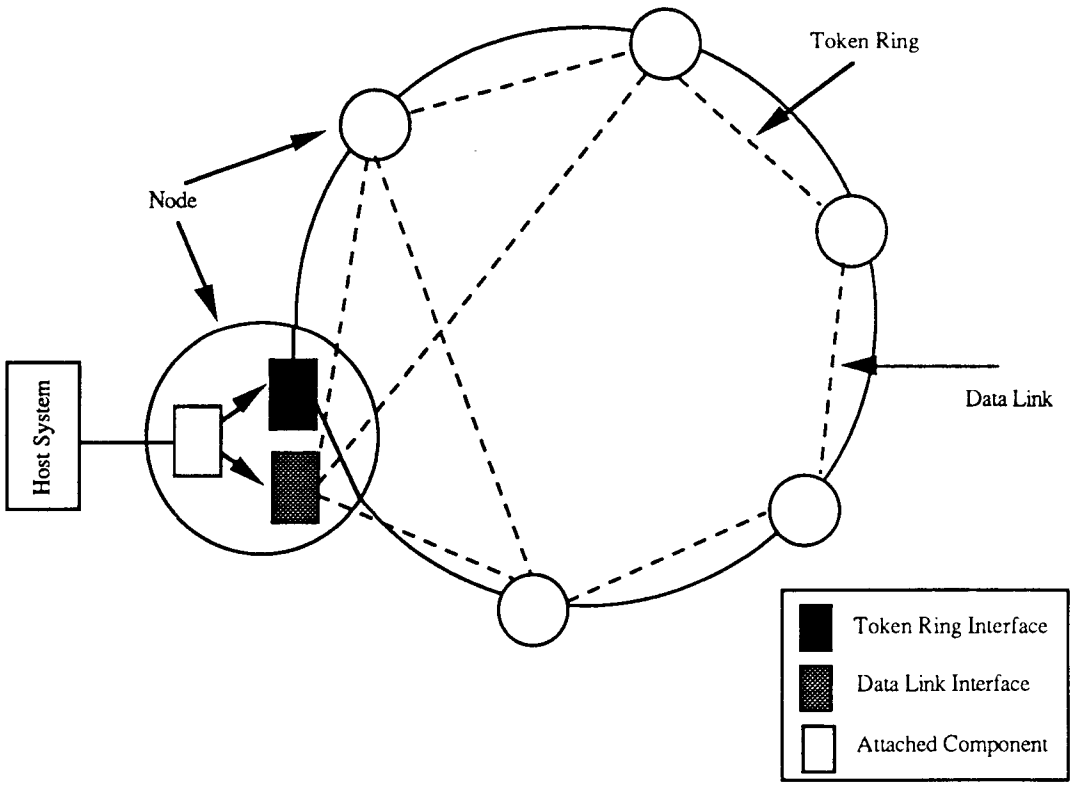


Figure 3.2 Network Interface Unit from the Network's View.

In the data link interface, the switching device is involved in connecting the incoming links to the outgoing links at an intermediate node. It allows the transmission or reception of a message over all available links at the source or destination. A message meant for a node is passed on to the data link interface receiver and stored in the receive buffer. The data link interface transceiver allows the transfer of a message from the transmit buffer to the data links via the switching device. The data link interface controller supervises/manages the activities of the data links and switching device, updating tables and maintain interface with the attached component and token ring interface.

Within the token ring interface, the receiver checks for a free token frame or the message frame addressed to it. If the message is addressed to the node, it copies and forwards the message; else it simply forwards the message. The token ring interface transmitter on reception of free token, sends a data or control message over the ring. The token ring interface controller supervises/manages activities associated with token ring protocol, allows the updating of tables and maintain interface with the attached component and data link interface.

The attached component interface controls the exchange of data between the host computer and the NIU. It is involved in routing decisions based on length, priority and destination information. It also makes decisions on learning mode and fault tolerance routing. It acts like a manager. After getting the requirements and analyzing the conditions it delegates the responsibility for transmission to either the data link interface or the token ring interface.

3.3 Organization of the Network Interface Unit

The NIU is to be designed such that it can work independently of the host processor. The maximization of performance and network efficiency can be thus

achieved. Since hybrid meshnet is still in the development stage, providing the unit as a programmable structure will allow flexibility in changing its protocols and associated functions. More complicated fault diagnosis and error correcting algorithms can be implemented, if desired. It will allow the 'user' to have control of some of the important parameters. Making the unit programmable will also make the design of control units simpler and less expensive. We want the NIU portable over a wide range of platforms. It is desirable to ease the task of redesigning the interface for different computers. The redesign work will be minimized if the bus dependent parts are separated from the rest of the system.

Design of the network interface unit is complicated by the fact that the communication is asynchronous and control is distributed. There can be multiple transmissions taking place at the same time on the network. Each node can either be a source or a destination and/or an intermediate node for one or more of these simultaneous transmissions.

Figure 3.3a shows the shared bus configuration in which all the activities of the network interface unit will be performed by a single processor. Communication with the host processor is via the shared memory. The communication will involve the transfer of commands and data between the two processors. As explained earlier, the NIU will have to perform many asynchronous activities and the single processor will not be able to handle them, especially when our objective is to design a high performance programmable unit. Also handling of each interface (token ring and data link) is rather work intensive and a well specified task, which makes each unit a natural candidate for an individual module.

Figure 3.3b shows the configuration with multiple processors forming the network interface unit. Communication with the host processor is still via the shared memory.

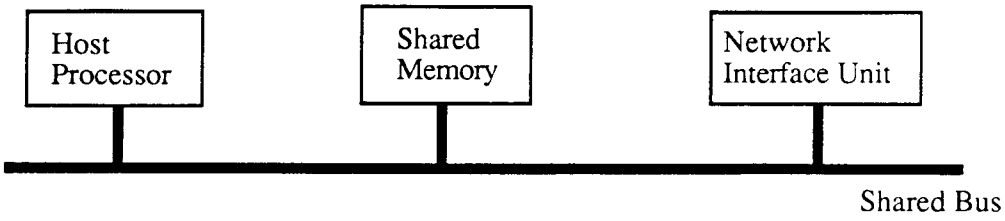


Figure 3.3a Shared Bus Configuration (Single Processor NIU)

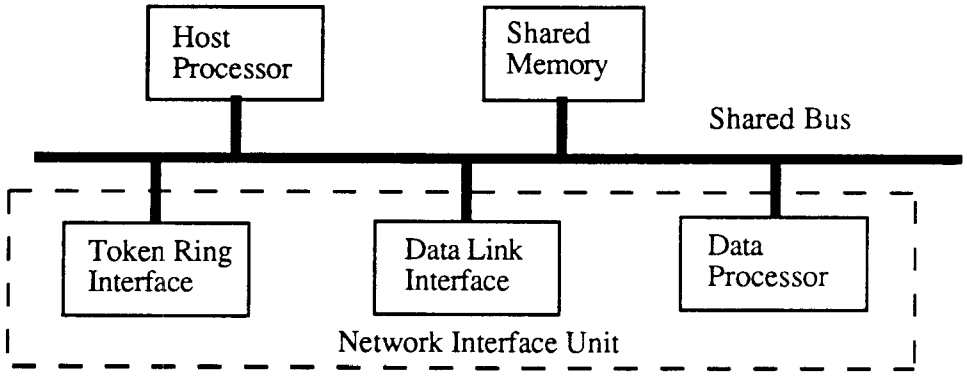


Figure 3.3b Shared Bus Configuration (Multi-Processor NIU)

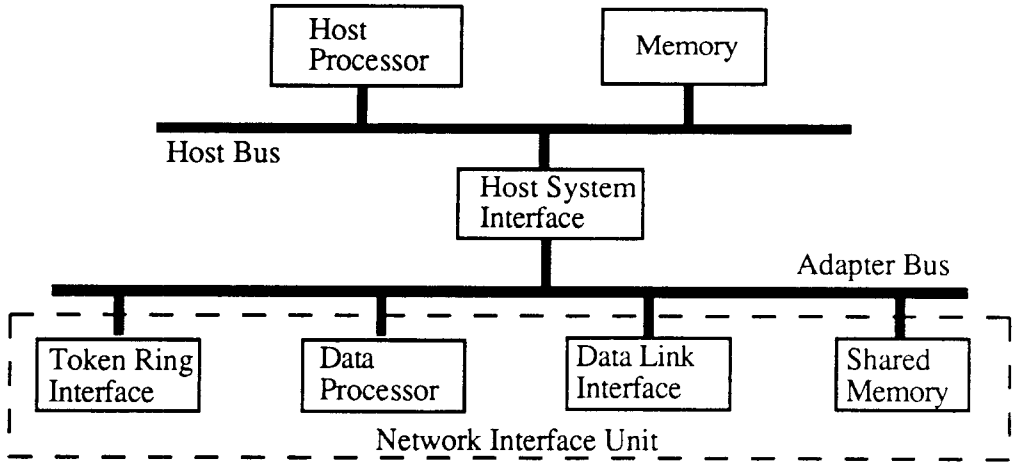


Figure 3.3c Multi-Processor NIU with Adapter Bus.

Note here that all processors share the same bus and memory. This organization does not quite allow us to achieve our objective of modularity. The design of the NIU will be host specific in this case, as each of the processors will be governed by the timing and control requirements of the host processor. Same units can not be ported to a different platform without modifications in the design of *each* processor.

Figure 3.3c shows an alternative configuration in which the NIU is separated from the host system via the Host System Interface. The NIU consists of three processors representing the functional units described in section 3.2. The processors share the same bus and communicate via the shared memory. However, communication to the host processor is via the Host System Interface using control signals. The Host System Interface manages the timing and control signal conversions between the two systems. The NIU can be considered as a sub-system. Dependency on the host is now reduced to the Host System Interface and can be simple. This will allow multiple platforms to be supported with minimum modifications. Using multiprocessors should prevent any performance bottleneck. The unit will allow most of the functionalities and parameters to be programmable with the expansion capabilities.

3.4 Modifications

Protocols and working of the Hybrid Meshnet is dealt with in [1]. However while attempting to give the architectural specification for the NIU, it was found that some modifications in the proposed protocols are required for the proper functioning of the network. Some of the changes are to maintain compatibility with the existing IEEE 802.5 standard for token ring. Some of the current conditions [1] and proposed modifications are:

1. *In the case of flood routing the message is passed from one node to the other over all available links without checking the destination address. This is done until the message reaches the destination.*

This can result in a congestion problem. The messages may not find a path to the destination and may move from one node to another before being removed. In order to avoid this, the control field in the data link frame is provided with a hop count. At each node the hop count is incremented and checked. If the hop count is less than the set limit, the message is forwarded over the links; if it is greater it is removed. The extra delay is introduced due to the requirement of checking the hop count but avoids messages unnecessarily loading the network.

2. *The path request control message assumes that the destination is always free to accept messages.*

In the case of multiple transmissions over the network, it may happen that the destination is busy as a source or a destination to another node. It may not then be ready to accept the data messages from the given node. The message transmitted thus will not be accepted at the destination. In order to avoid this, a busy bit is added in the control field of the frame. If the destination is not ready to accept messages its token ring interface will append the busy bit. Transmission will be then kept pending until the destination is free. The free status of the destination can be determined by monitoring the control messages. The source unit stores the busy destination and checks a control message every time it is received. If either the source or destination field matches the busy destination address it assumes that destination is free and attempts to transmit again.

3. There are some minor changes proposed to the token ring protocols suggested for hybrid meshnet in [1]. These are based on IEEE 802.5 standard [3] :

(1) The Frame check sequence (FCS) should be 4 octets instead of 2 octets.

(2) The control field will have Access Control (AC) and Frame Control (FC) fields. The token bit and other control bits' positions are modified to conform to the standard.

(3) Addition of Frame Status (FS) field. This will allow an acknowledgement of the message being recognized and copied by the destination to the originating node. The token ring data/control frame is as shown.

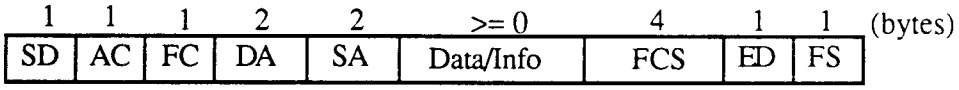


Figure 3.4 Token Ring Data/Control Frame.

4. The connection acknowledgement (CACK) frame for the data link interface is modified to include the starting flag (SFLAG), ending flag (EFLAG), control frame, destination address (DA) , source address (SA) and cyclic redundancy check (CRC). This results in a uniform protocol as all the messages have the similar format. The decoding unit at each link is thus simplified.

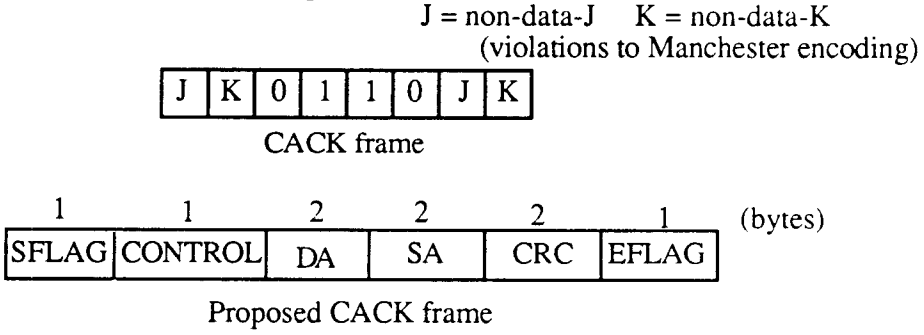


Figure 3.5 CACK Frames.

While proposing the design features for the Hybrid Meshnet, some assumptions are made:

1. A node can not be a source or a destination to more than one transmission at the same time. However it can be an intermediate node for many transmissions, constrained only by the number of links available as per the implementation.
2. Each node maintains routing information related only to itself.
3. The higher layers have the knowledge of addresses of other nodes on the network. At the time of initialization, this information is passed to the LLC layer.

CHAPTER 4
DESIGN FEATURES OF THE NIU

In this chapter, we will take a detailed look at the design features of the Network Interface Unit for Hybrid Meshnet. The organization of the NIU is similar to the one suggested in Figure 3.3c.

The network interface unit performs all time critical functions independently of the host processor, and maximizes performance and network efficiency. Various MAC and LLC layer functions including routing, framing, preamble generation. destination address checking, CRC generation and checking are carried out by the NIU.

4.1 Organization of the Network Interface Unit

In this section, we will look at how the different blocks of the NIU interface with each other as well as with the host processor. As shown in Figure 4.1, the Host System Interface, the Data Processor, the Token Ring Interface and the Data Link Interface together form an integrated Hybrid Meshnet node.

The data processor, token ring interface and data link interface, each include a local CPU with separate instruction and data space. The subsystem imitates a Multiple Instruction Multiple Data (MIMD) computer with each processor executing its own instructions from its own memory. Communication between the processors is through a set of interrupts and shared memory via the 16-bit shared address and data bus.

The host system interface provides the interface between the host system and the NIU. It will allow for timing and control signal conversions between the two systems. The data and control messages from the host memory are transferred to the local memory via DMA. The direct memory access in this case differs from conventional DMA in that the data transfers occur from memory of one bus to memory of the second bus. This is

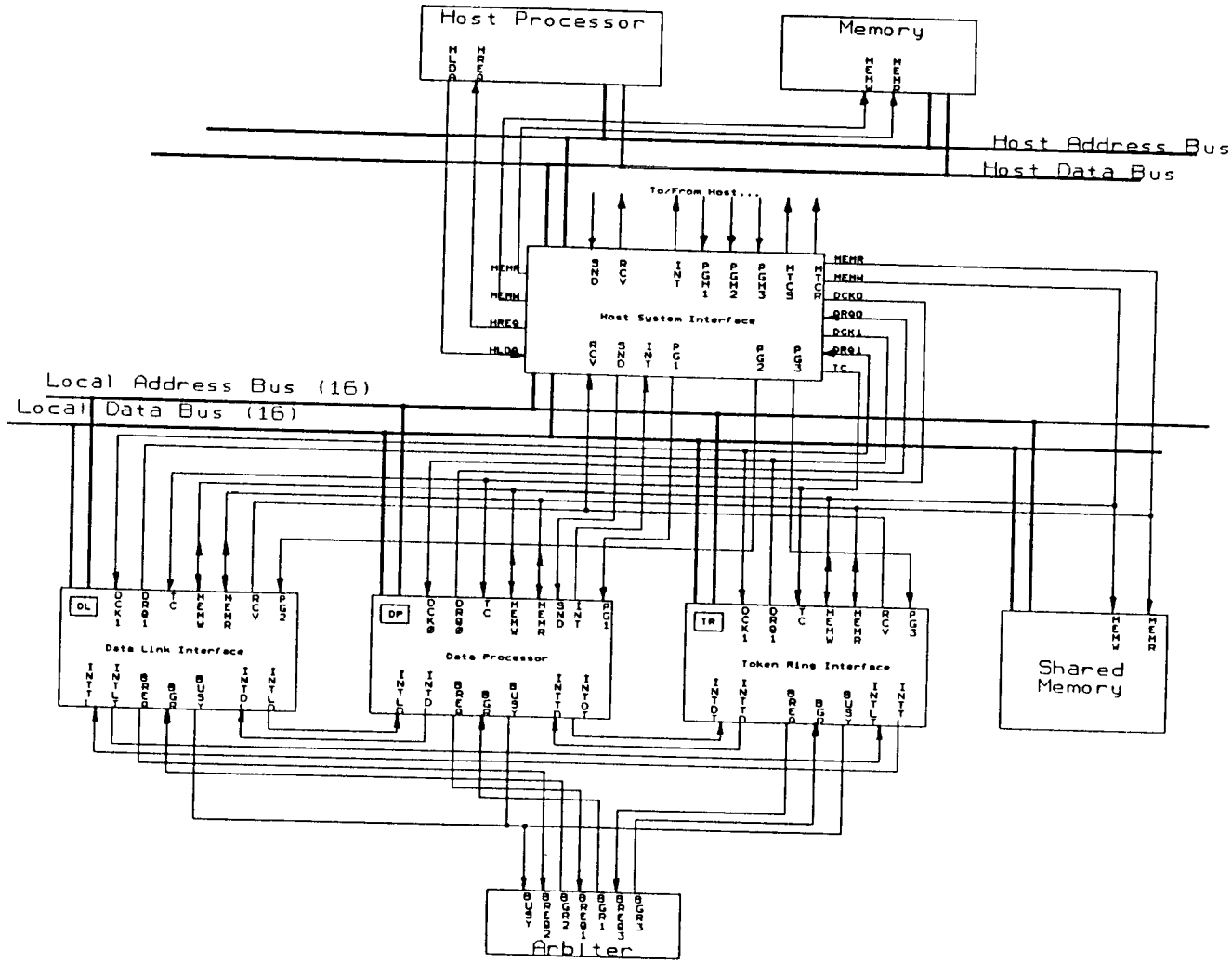
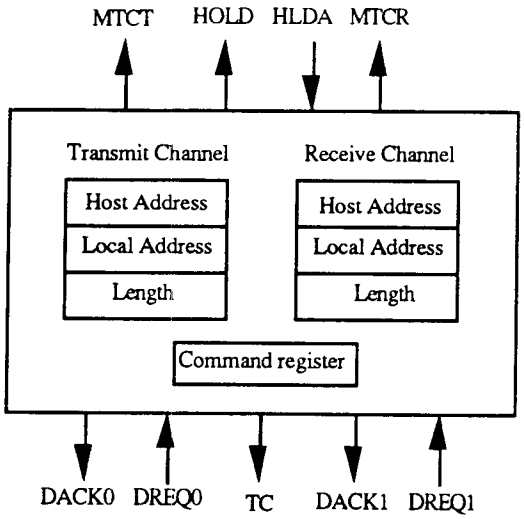


Figure 4.1 The Network Interface Unit

in direct contrast to the conventional DMA controllers which use shared memory on single bus. The two buses are independent of each other in all respects, including timing, leading to the asynchronous transfer of data. The host system interface provides for the receive and transmit channels with the registers for host memory address, local memory address and length as shown in Figure 4.2.



HOLD - Hold request to the host processor.
HLDA - Hold Acknowledgement from the host.
MTCT - Message transfer complete for transmission.
MTCR - Message transfer complete for reception.

DREQ0, DREQ1 - DMA request by an interface for transfer of data from/to host respectively.
DACK0, DACK1 - Acknowledgement to an interface for transfer of data from/to host respectively.
TC - Transfer complete.

Figure 4.2 Host System Interface for DMA.

The data processor (DP) obtains the control information for the message to be transmitted from the host memory and makes routing decisions accordingly. Each decision is also based on the previous history of routing to a node, the current state of the node and the current state of the network. Based on the decision, one of the many possible algorithms is followed, resulting in a successful connection and transfer of the message to the destination node. The data processor maintains routing and decision tables, and provides a snap shot of the activities on the network relative to itself. Details of the data processor will be explained later in Section 4.3.

The token ring interface (TRI) performs all activities required for communication over the ring. It accepts commands from the data processor through the shared memory and implements the modified token ring protocol. The LLC frames are received from the

host memory via the host system interface using DMA. Details of the token ring interface will be explained later in Section 4.4.

The data link interface (DLI) based on the control information from the token ring interface and data processor, performs the necessary functions to carry out the transmission of the message over the link(s). Details of the data link interface will be explained later in Section 4.5.

All the communication between the three processors of the subsystem takes place via shared memory. The shared memory stores control messages from the host to DP, control and data messages for the modules and routing tables required for fault routing. Under the current implementation, a 8-16 Kbyte memory should be more than sufficient. The memory is partitioned as shown in Figure 4.3, with each part acting as a mailbox for a particular control or data information.

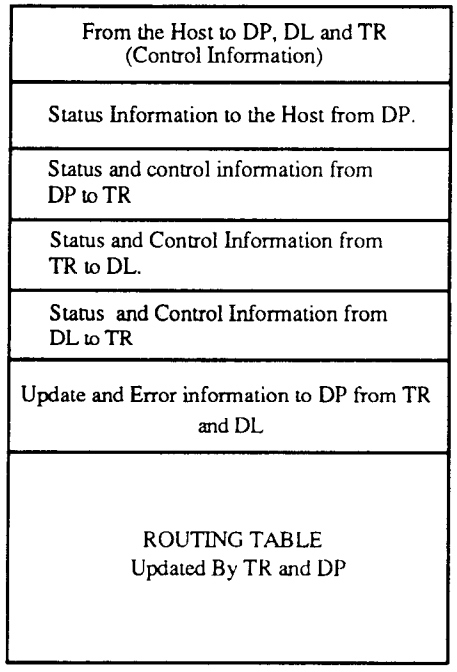


Figure 4.3 Shared Memory Organization

It is necessary to avoid the problem of bus contention in the shared bus configuration. An arbiter controls the access of the shared bus. When the use of the bus is required, the requesting units will send BREQ signals to the arbiter. If the BUSY signals are negated (indicating a free bus), the arbiter resolves the requests based on priority and sends a BGR signal to the appropriate unit, which can then use the bus exclusively. The unit asserts the BUSY signal, indicating that the bus is not free for use by other units. Generally, a rotating or a fixed priority scheme can be used to resolve simultaneous requests for the bus by more than one interface. In hybrid meshnet, each of the interfaces is driven by the interrupts from the other two interfaces, resulting in a kind of a master-slave relationship. This results in little simultaneous need for the bus and hence such a priority scheme may not be required.

The network interface unit provides the functions required to implement a Hybrid Meshnet station. It features the use of user configurable parameters. Configurable hardware parameters enable the user to tailor and thus optimize the communication system operation for different network configurations and applications. These parameters are passed to the processors under the supervision of the host processor using PG1-PG3 interrupts (refer to Figure 4.1) at the time of initialization.

A communication network is operated, controlled and managed through control programs. These programs are concurrently executed in the three units. Except for the time critical and non-varying functions, all the functionalities of the NIU are programmable. This will provide for tremendous flexibility in the design, validation and standardization of Hybrid Meshnet. A general purpose interrupt (INT) from the DP to the host will serve to provide additional services to the higher layers. Its functionality, however, is not defined at present.

There are two independent media over which the data transmission can take place : token ring and data links. In order to provide functionalities to both of these media in the

MAC layer, the layer is partitioned. Token ring interface and data link interface map to these two parts. Besides the MAC functions these two interfaces also implement some LLC functions like establishing connections to other nodes in the network. Other LLC functions like routing takes place in the data processor while still other functions like forming LLC Protocol Data Units (LPDU) are carried out in software by the host system. The mapping of the MAC and LLC layers to the NIU is as shown in Figure 4.4.

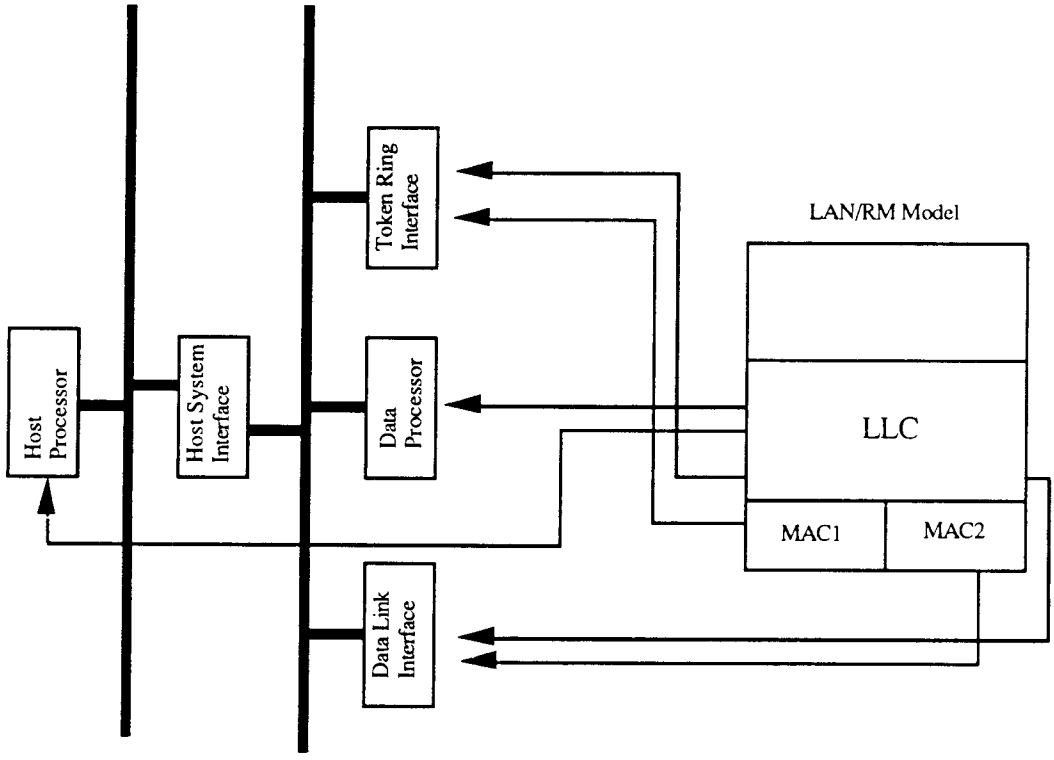


Figure 4.4 Relationship between LAN/RM Model and the NIU.

General flow of control and data messages is as shown in Figure 4.5. All three interfaces work asynchronously with each other. The communication between them is via shared memory and interrupts. *The host processor is relieved of any processing relating to the communication over the hybrid meshnet.*

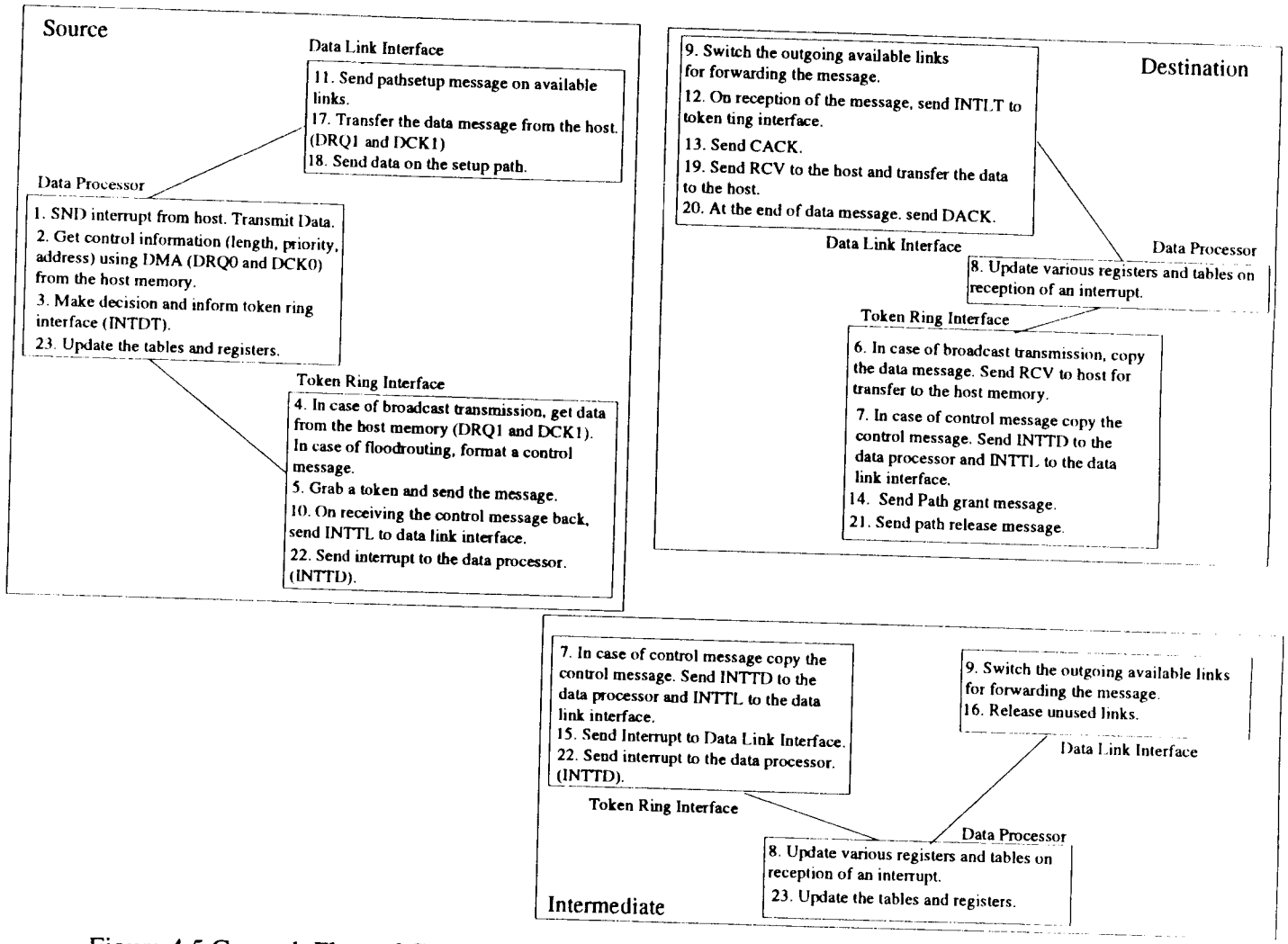


Figure 4.5 General Flow of Control and Data in NIU for Broadcast Transmission and Flood Routing.

4.2 Design Strategies for the Processors

The network interface unit presented in the earlier section will be a multiprocessor system. In following sections we will look into the requirements and design of each of the sub-units.

The purpose of this section is to present the processor architecture which forms the basis for all the three units. Having a common architecture will simplify and reduce the design effort. The details specific to each unit will be discussed in later sections.

The requirements for the processors forming the network interface unit are high performance and simplicity. A high performance network interface unit ensures that the unit will work efficiently and that any performance bottleneck in the system is not the result of unit itself. Simplicity of the design is associated with simple instruction sets, few addressing modes and uniform opcode formats, resulting in simpler control units.

Early analysis of the requirements indicated that processing involved at each processor is not very complex. Simple instructions can provide most of the functionalities. Additional hardware and instructions will be required to carry out specific functions. Looking at the requirements, the RISC architecture seemed to be a natural choice. The basic block diagram for the architecture is shown in Figure 4.6. The detailed architecture (excluding the control unit) is as shown in Figure 4.7. It adheres to the requirements of RISC : small set of instructions, few addressing modes, fixed instruction format, instruction executed in one cycle, only load/store for memory access and pipelined structure. The architecture has a few ideas borrowed from 'Computer Architecture- a quantitative approach' by John Hennessy and David Patterson.

The processor has a separate instruction and data space with the instruction being 32-bits long. The proposed RISC architecture is a 3-stage pipeline structure with a 4-phase clock.

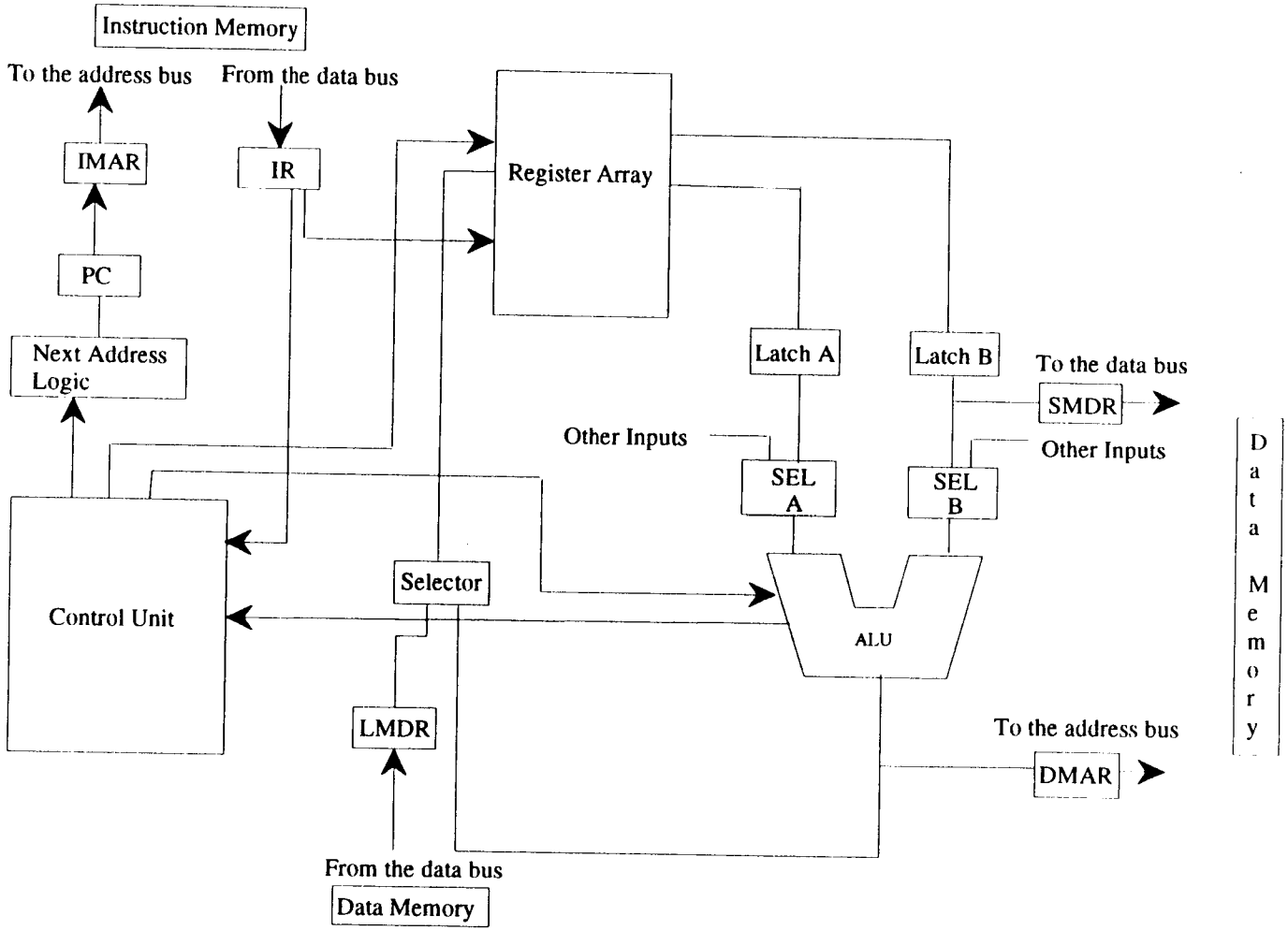


Figure 4.6 Simplified Block Diagram for the Basic Architecture.

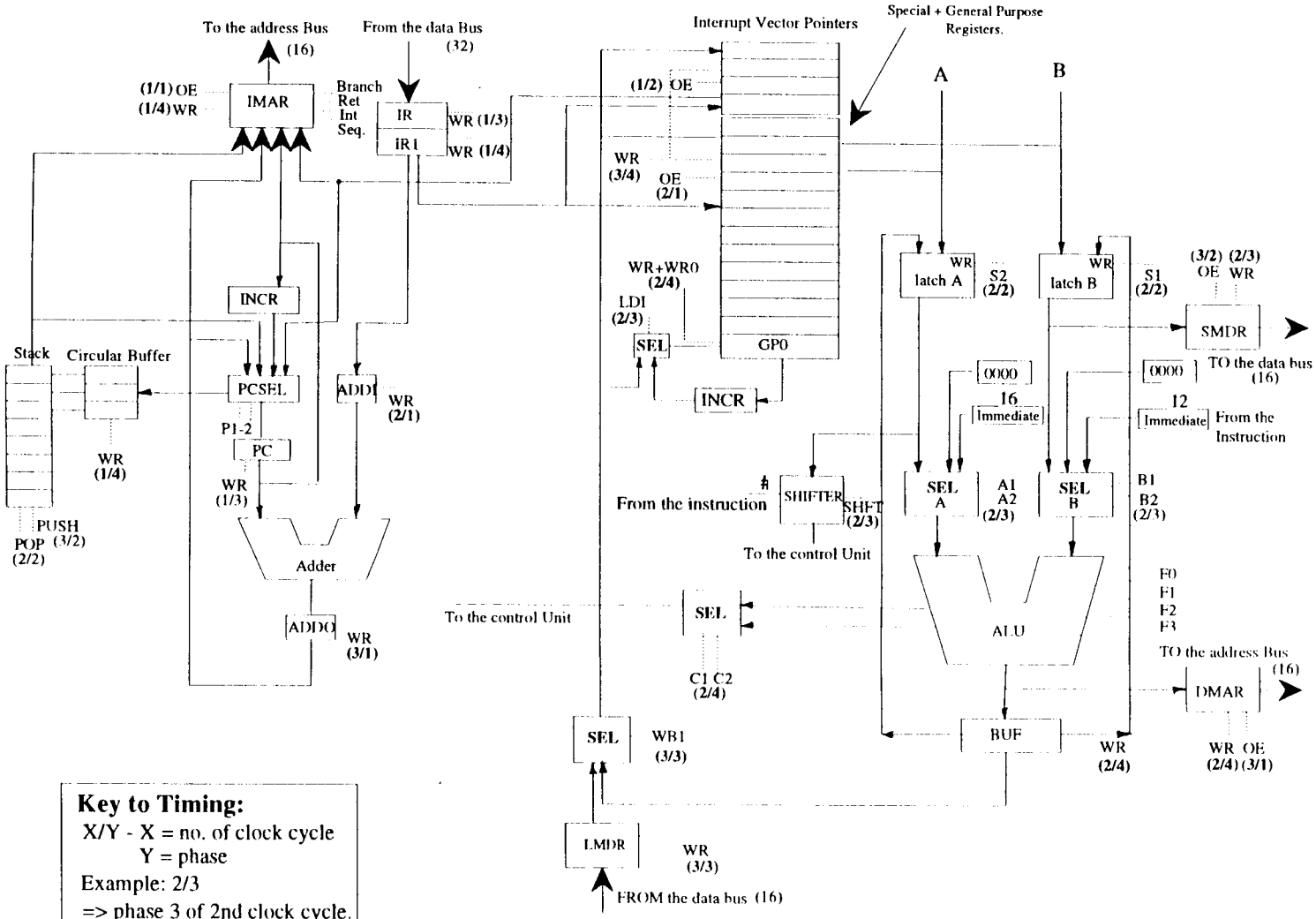


Figure 4.7 Basic Architecture of the Processors.

Refer to Figure 4.6. The first stage (IF) involves the fetching of an instruction from the memory. The instruction memory address register (IMAR) places the address of the next instruction on the bus. By the end of the first stage, the instruction is available on the data bus and is latched into the instruction register (IR).

In the second stage (ID/EXE), the instruction is decoded and appropriate control signals are generated by the control unit. The arithmetic logic unit (ALU) carries out the required function on the input data. For the load/store instructions, the ALU output is a memory address. The data memory address register (DMAR) holds this output for the next stage.

The result is written back to the specified register in the register array during the final stage (ME/WB). In case of load instructions, the data from the address specified in DMAR is loaded into the load memory data register (LMDR). For the store instructions, the data from the store memory data register (SMDR) is placed on the data bus.

Events on each stage of the pipeline due to different instruction types is shown in Figure 4.8. Note that this indicates only the general flow and does not show all the operations.

The timings for each of the operations is shown in Figure 4.7. The key to the timings is also shown. Different clock cycles in the diagram are in reference to a single instruction. The three clock cycles represent the three stages in an instruction execution.

Every pipe stage is active on every clock cycle. This means that all the operations in a stage must complete on every clock cycle. It is necessary to ensure that no two stages require access to the same resource at any given time. The resource table of Figure 4.9 indicates that there are no resource contentions in our processor architecture. The resource table is derived on the basis of timing information as in Figure 4.7 and resource usage by each type of instruction as shown in Figure 4.8.

Stage	ALU Instruction	Load/Store Instruction	Branch Instruction
IF	IR <- Mem[IMAR]; PC<- PC +4;	IR <- Mem[IMAR]; PC<- PC +4;	IR <- Mem[IMAR]; PC<- PC +4;
ID/EX	latch A<-bus A latch B<-bus B IR1<-IR ALUoutput<-A op B BUF<-ALUoutput	latch A<-bus A latch B<-bus B IR1<-IR SMDR<-latch B ALUoutput<- A op B DMAR<-ALUoutput BUF<-ALUoutput	ADDI<-immediate field of IR. cond<-A op 0 or cond<-A shift #
ME/WB	Dest. Reg.<-BUF	Mem[DMAR]<-SMDR or LMDR<-Mem[DMAR] Dest. Reg.<-LMDR	if (cond) PC<-ADDO

Figure 4.8 Events on Every Stage of the Pipeline.

In RISC architecture, there are situations called hazards that prevent the next instruction in the instruction stream from being executed during the designated clock cycle. They may be due to resource conflicts, dependence of instruction on the results of previous instructions or pipelining of branch instructions. These are termed as structural, data and control hazards respectively. They degrade the performance and must be avoided.

Structural hazards are prevented by designing a proper pipeline structure. As shown earlier, there are no resource conflicts in the proposed architecture. Data hazards are overcome by including architectural modifications. A single buffer at the output of ALU is provided. The buffer stores the result of previous instruction execution and, if required, feeds back the data to the ALU at the next cycle under the action of control unit. The number of buffers depend on the number of stages in the pipeline. With little additional complexity for the control unit, data hazards are avoided. Control hazards can be overcome by the concept of a branch-delay slot. The scheme employs

Control Signals (refer toFig 4.7)

X - active control signal

Stage	IMAR WR	IMAR OE	IR	IR1	PC WR	ADDI	ADDO	REG WR	REG OE	S2 A	S1 B	A1 A2	B1 B2	SHFT	BUF WR	WB	C1-C2	DMAR OE	DMAR WR	SMDR WR	SMDR OE	LMDR
IF	X	X	X	X	X																	
ID/EXE						X			X	X	X	X	X		X							
ME/WB							X	X								X						

ALU Insructions

Stage	IMAR WR	IMAR OE	IR	IR1	PC WR	ADDI	ADDO	REG WR	REG OE	S2 A	S1 B	A1 A2	B1 B2	SHFT	BUF WR	WB	C1-C2	DMAR OE	DMAR WR	SMDR WR	SMDR OE	LMDR
IF	X	X	X	X	X																	
ID/EXE						X			X	X	X	X	X						X	X		
ME/WB							X	X								X		X			X	X

Load/Store Instructions

Stage	IMAR WR	IMAR OE	IR	IR1	PC WR	ADDI	ADDO	REG WR	REG OE	S2 A	S1 B	A1 A2	B1 B2	SHFT	BUF WR	WB	C1-C2	DMAR OE	DMAR WR	SMDR WR	SMDR OE	LMDR
IF	X	X	X	X	X																	
ID/EXE						X			X	X	X	X	X	X			X					
ME/WB							X															

Branch Instructions

Figure 4.9 Resource Allocation for Different Instruction Types.

rescheduling of instructions so that the instructions which are independent of the decision are executed in cycles immediately following the branch instructions. If implemented in software, the compiler or the assembly programmers should take care to avoid such hazards.

Instruction execution is initiated through a set of interrupt calls. The interrupts can be external (from an other interface) or internal. When an interrupt occurs, the return addresses are pushed on the stack and the service routine is executed. It is necessary to save and restore more than one program counter (PC) value to ensure that the processor has been brought back to the state it was before the interrupt occured. To understand this requirement, let us take a look at the following steps in execution. Assume that only the last program counter value is pushed onto the stack.

```
BEQZ r5,immediate     /* Branch to immediate address if r5 =0 */
ADD r3,r4,r2          /* r5 := r2 + r4 */
SBB r1,r2,r3          /* r1 := r2 - r3 */
```

After three clock cycles, the instructions are in the following stages.

```
SBB r1,r2,r3              -- IF stage.
ADD r3,r4,r2              -- ID/EX stage.
BEQZ r5,immediate         -- ME/WB stage.
```

Note that the execution of ADD and SBB instructions have been rescheduled to avoid the control hazards. These instructions are executed irrsepective of the result of the branch instruction. Suppose the result of a branch instruction is true. The program counter at this stage will have the address of the branch as it is required to fetch the next instruction from the branch address. If an interrupt occurs at this point, the SBB and ADD instructions will not be completed. These instructions should be fetched again from the memory once the interrupt routine is serviced. However, the return address in the program counter corresponds to the branch address. The program execution will

continue from the branch address resulting in ADD and SBB instructions not being executed at all.

By storing the last three program counter values, it is possible to bring the processor back to the original state. For the above example, it will allow us to fetch the ADD and SBB instructions after the interrupt routine. This is done using a combination of a circular buffer and a stack. A circular buffer holds the last three values of the PC. When an interrupt occurs the values in the circular buffer are pushed on to the stack under the action of the control unit. They are popped back from the stack when the processor finishes the interrupt routine.

This forms the basic processor architecture. It provides a throughput of one instruction/cycle. The instruction formats, discussed in later sections, are designed such that the source and destination registers as well as immediate values are at the same positions in each instruction. This will allow for registers to be loaded before the instruction is decoded, resulting in performance improvement and simpler control units. The control unit will mainly have to decode the opcode to determine the type of instruction (which are few) and generate control signals accordingly.

Let us now look at each of the processors specifically.

4.3 Data Processor

Hybrid meshnet involves transmission of data over one of the two possible media : token ring or data channels. The routing decision to send data on one of the media rather than the other is made by the data processor. The data processor also initiates learning mode routing and fault diagnosis routing whenever appropriate. The working of the data processor can be understood from the Figure 4.10.

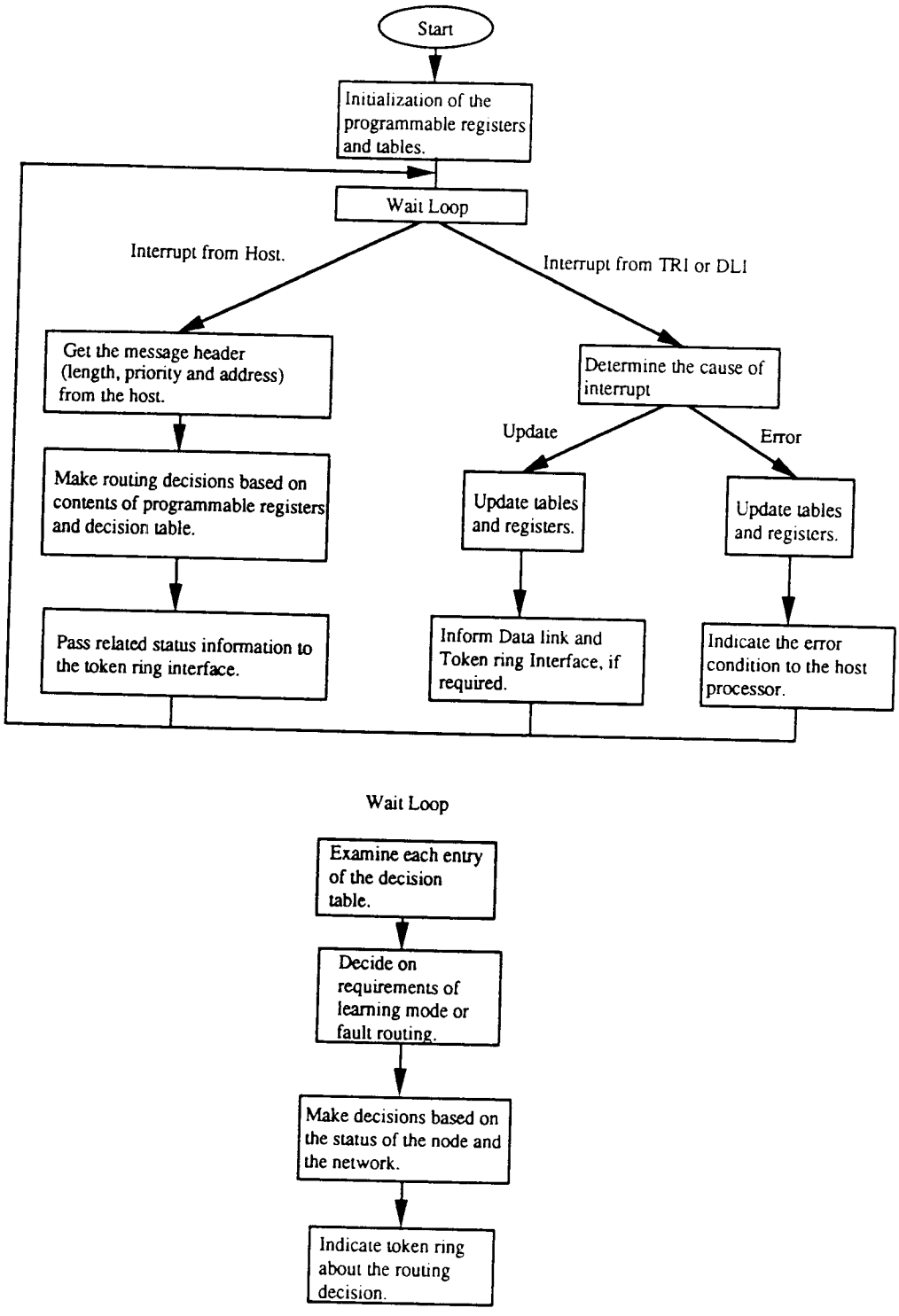


Figure 4.10 Working of the Data Processor.

Routing decisions made by the data processor are based on a number of factors : length and priority of the message, the destination address, availability of the node and the previous routing history to the node. Initiation of learning mode routing or fault routing depends on the state of the network, necessity of learning mode routing to a node and the previous history of routing to a node. This will be explained later in the section.

The basic processor architecture of the data processor is as explained in the earlier section. The register organization for the data processor is as shown in Figure 4.11. This represents the register array shown in Figure 4.6. The service routines' execution is initiated by a set of interrupt calls. Interrupt vector pointers provide the address of the interrupt routines. These addresses are programmed during initialization. The 24 general purpose registers are divided into 3 banks of 8 registers each. The processor may switch register banks under program control between the interrupt routines. Each of the interrupt priority levels (0 and 1) is assigned a separate register bank. The third register bank is used by non-interrupt driven service routines. This relieves the processor from saving and reloading registers to provide the necessary register space. The register array will have 2 read ports and 1 write port. All ports can be active concurrently without conflict.

Priority levels for the interrupts can be zero or one. Suppose an interrupt Y of priority 0 is being serviced. If the interrupt X of priority 1 occurs during this time, the register banks are switched and the interrupt routine for X is executed. At the end of the routine, the register bank is switched back to the node corresponding to Y and the suspended routine continues. The interrupt having the same level as the one being serviced is kept pending until the present routine is completed.

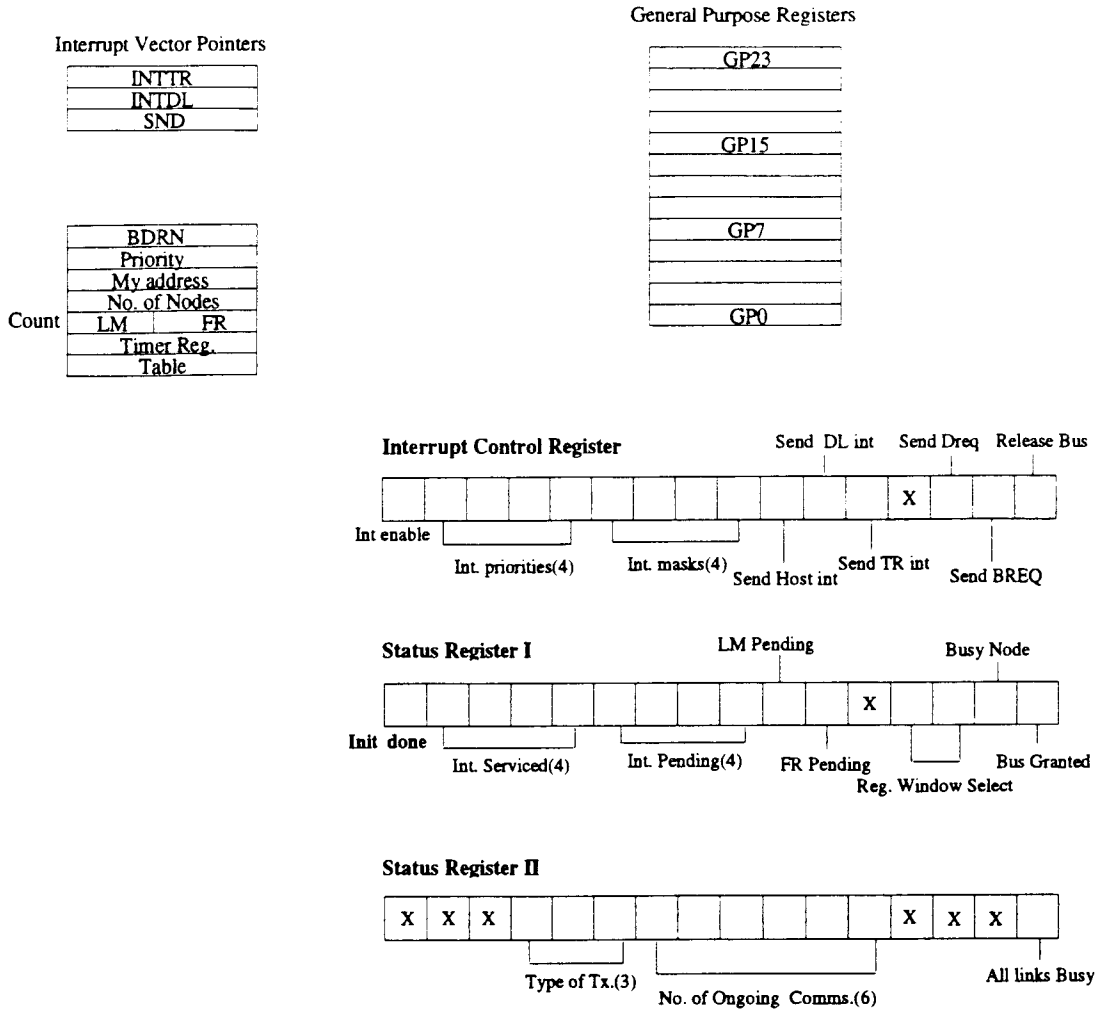


Figure 4.11 Programmable Registers for the Data Processor

Destination	Init.	Busy	LM Count	FR Count	Type of Tx

Figure 4.12a Decision Table (256x32)

Destination	Address in Mem.

Figure 4.12b Routing Table (256x32)

Interrupt priorities and mask information are kept in the interrupt control registers. The status register keeps track of the pending and serviced interrupts. The interrupts as well as control signals to the other units are initiated under program control.

The routing decisions are made by the processor based on the contents of a few programmable registers. The messages having length smaller than the value in broadcast data transmission reference number (BDRN) and priority greater than the value in PRIORITY are sent over the token ring. Other messages are sent over the data links.

The data processor is also equipped with the decision and routing tables (Figures 4.12a-b). The tables are essentially content addressable memory (CAM). The addressable contents in both cases being the node addresses. The decision table keeps track of the busy nodes and the failed routing attempts to a particular node. Based on this table, the data processor makes a decision to initiate flood routing. If the node is busy, the transmission is delayed. If the number of failed routing attempts exceeds the threshold set by the 'user', flood routing is not initiated.

Depending on the Init flag (which indicates whether the path to the node is known/unknown) and the learning mode (LM) count (have enough attempts been made to learn the path to a node?), the data processor initiates the learning mode routing during periods of no load on the network. The information learned during the learning mode can be accessed via the routing table. The routing table keeps track of the addresses in shared memory where the routing information to a particular node is stored. In case of fault diagnosis routing, the routing table is used to get these addresses.

Instruction format and instruction types are as shown in Figure 4.13 and Figure 4.14. As can be seen, the instruction requirements for the data processor is simple and minimal. No extra hardware is required to execute this set of instructions.

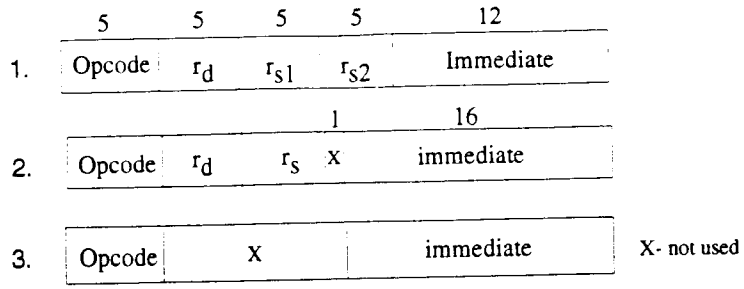


Figure 4.13 Instruction Format for the Data Processor.

Instruction	Type	Function
MV reg.,reg.	2.	Move data from register to register
MVI reg., immediate	2.	Move an immediate value to register.
ADD reg.,reg.,reg.	1.	Add a reg. to reg. . Store the value in the third reg.
SBB reg.,reg.,reg.	1.	Subtract a reg. from a reg. Store the value in third reg.
ADI reg., reg., immediate	2.	Add immediate to reg. Store the value in third reg.
SBI reg.,reg.,immediate	2.	Subtract immediate to reg. Store the value in third reg.
AND reg.,reg.,reg.	1.	AND the contents of two regs. Store the value in third reg.
OR reg.,reg.,reg.	1.	OR the contents of two regs. Store the value in third reg.
XOR reg.,reg.,reg.	1.	XOR the contents of two regs. Store the value in third reg.
ANI reg.,reg.,immediate	2.	AND the immediate with reg. Store the value in third reg.
ORI reg.,reg.,immediate	2.	OR the immediate with reg. Store the value in third reg.
XRI reg.,reg.,immediate	2.	XOR the immediate with reg. Store the value in third reg.
SHL reg.,reg.,immediate	2.	Shift left the contents of reg. (No. of shifts = #immediate)
SHR reg.,reg.,immediate	2.	Shift right the contents of reg. (No. of shifts = #immediate)
LD reg. , [reg. + immediate]	1.	Load the data from memory to the specified reg.
LDI reg.,[reg]	1.	Load the data from memory and increment the memory add.
ST [reg. + immediate], reg.	1.	Store the data from the reg. into the memory.
BEQZ reg.,label	2.	Branch if the value of reg. = 0.
BNEZ reg.,label	2.	Branch if the value of reg. <> 0
STGE reg1,reg2,reg3	1.	Set the reg1 =1 if reg2 >= reg3
STLE reg1,reg2,reg3	1.	Set the reg1 =1 if reg2 <= reg3
STEQ reg1,reg2,reg3	1.	Set the reg1 =1 if reg2 = reg3
BSET reg.,#,label	2.	Branch if the #bit of the register is set.
BRSET reg.,#,label	2.	Branch if the #bit of the register is set.
JMP label	3.	Jump to the given address.
CALL label	3.	Call the routine.
RET	X	Return
NOP	X	No operation.

Figure 4.14 Instruction Set for the Data Processor.

The sample implementation shows that almost all the functionalities are programmable. The data processor functions are carried out through a set of interrupt calls.

- Data processor has various programmable parameters. The host system initializes the processor by sending a PG1 interrupt (refer to Figure 4.1).
- On interrupt from the host system, the message header is received and routing decisions are made.
- On interrupt from either token ring or data link interface, another routine updates tables and status contents of various registers to reflect the current state of the network.
- In a wait loop, it checks the decision table and initiates learning mode routing or fault routing, if required, under no load conditions on the network.

The assembly language implementation of the data processor functionalities is shown in the Appendix A. An attempt has been to incorporate most of the functionalities. The protocol is approximately 500 instructions long. Based on this implementation, it can be suggested that program memory of size 4 Kbytes will be sufficient. The memory can be on-chip or as an external ROM. The on-chip ROM will reduce the pin count, but on the other hand can not provide the same flexibility to modify the protocol as the external ROM implementation.

4.4 Token Ring Interface

The token ring interface (TRI) accepts the control information from the data processor and accordingly sends control or data messages over the ring. The interface implements all the MAC layer and few LLC layer functionalities. It provides for generation and distribution of token, frame address recognition, differential Manchester

encoding and decoding, CRC generation and checking. It also generates control frames for the data messages to be transmitted over the data links.

Figure 4.15 shows the logical operation of the interface. They function in two modes : listen and transmit. Stations are normally in the listen mode.

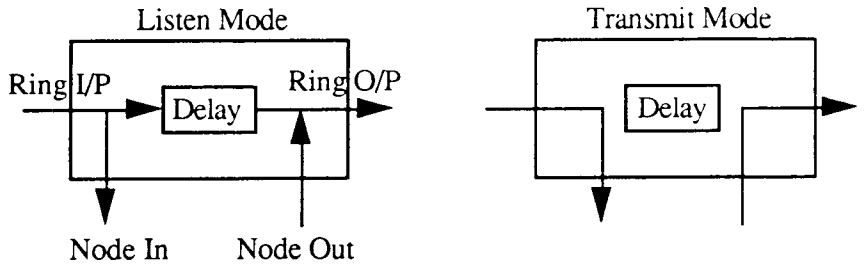


Figure 4.15 Logical Operation of the Token Ring Interface.

The basic block diagram of the TRI is as shown in Figure 4.16. The message from the ring arrives at the receive switch. It is copied and forwarded or simply forwarded depending on the type of message (control or data) and the destination address. The token ring processor manages the overall activities of the TRI. The processor is involved in initiating transmission, receiving messages, generating error and control messages, transfer of data messages between the host memory and FIFOs

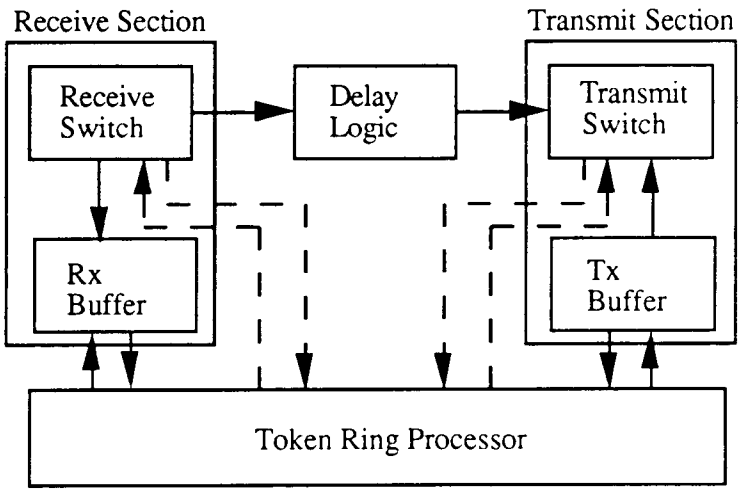


Figure 4.16 Basic Block Diagram of the Token Ring Interface.

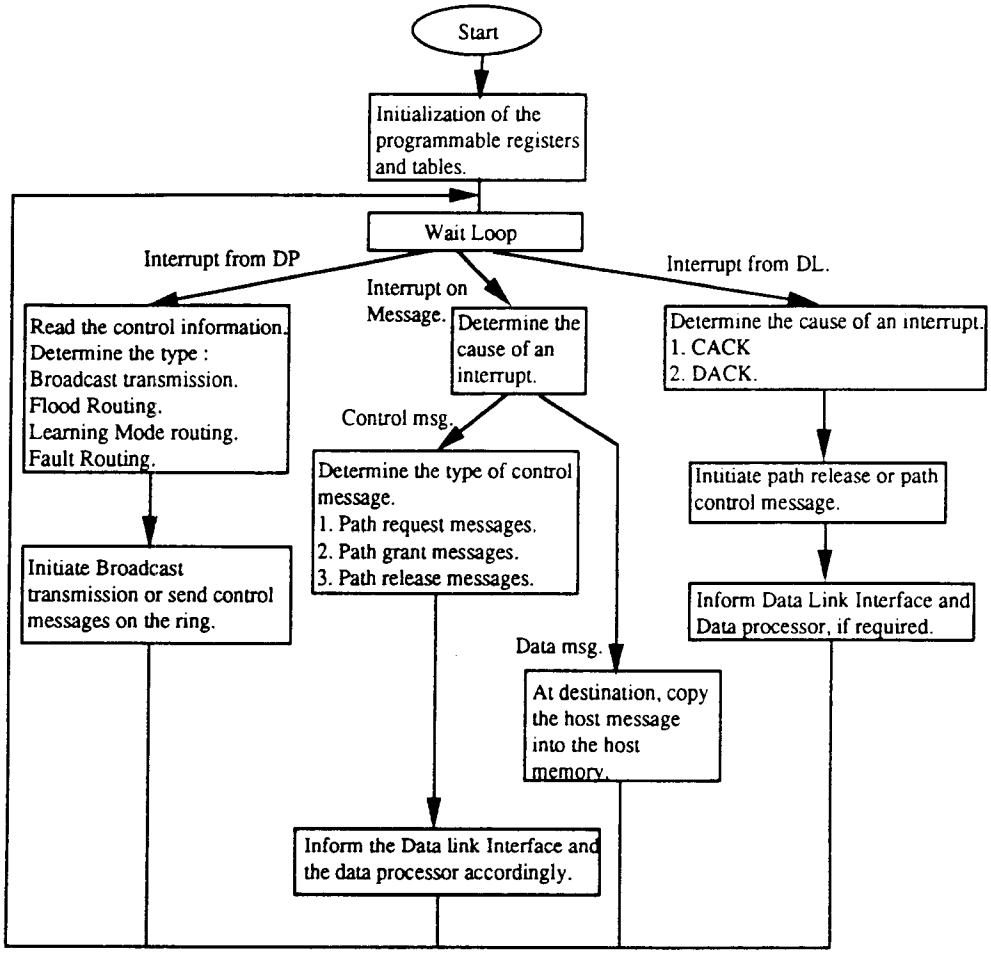


Figure 4.17 Working of the Token Ring Interface.

and synchronizing activities with the data link interface.

The delay logic essentially provides the delay required for the recognition of a free token, changing it to a busy token and appending the message at the source node. The delay logic provides a 1 bit delay at each node to carry out this action. This is the inherent latency associated with each node. In our implementation, this delay is programmable. The transmit switch will either forward the message (intermediate node) or initiate the message (source node). Working of the token ring interface in relation to the network interface unit can be understood from Figure 4.17.

The basic processor architecture for the TRI is as explained in section 4.3. The programmable registers available for the interface are shown in Figure 4.18. This represents the register array shown in Figure 4.6. The program execution is initiated through a set of interrupt calls. Interrupt vector pointers provide the address of interrupt routines. The interrupts can be from the data processor or the data link interface. Internally, interrupts are generated due to either a non-empty Rx FIFO or the timer count for 'timer, hold token' (THT) or 'timer, no token' (TNT) reducing to zero. The interrupt vector pointers are programmable at the initialization. The available interrupts can be prioritized (0 or 1) or masked by setting or resetting the specific bits in the interrupt control register (ICR). The interrupts and the control signals to the other interfaces are generated under program control by writing into the ICR or the control register (CR). The control and status registers provide the snapshot of the overall working of the TRI.

There are 3 banks of 8 registers each. The register banks are switched depending on the priority of the interrupt. SRC and DEST registers provide the control and status information for the Rx and Tx sections. The SRC register is provided with a flag which is a result of the modified protocol. When this flag is set, the TRI initiates transmission without waiting for the free token.

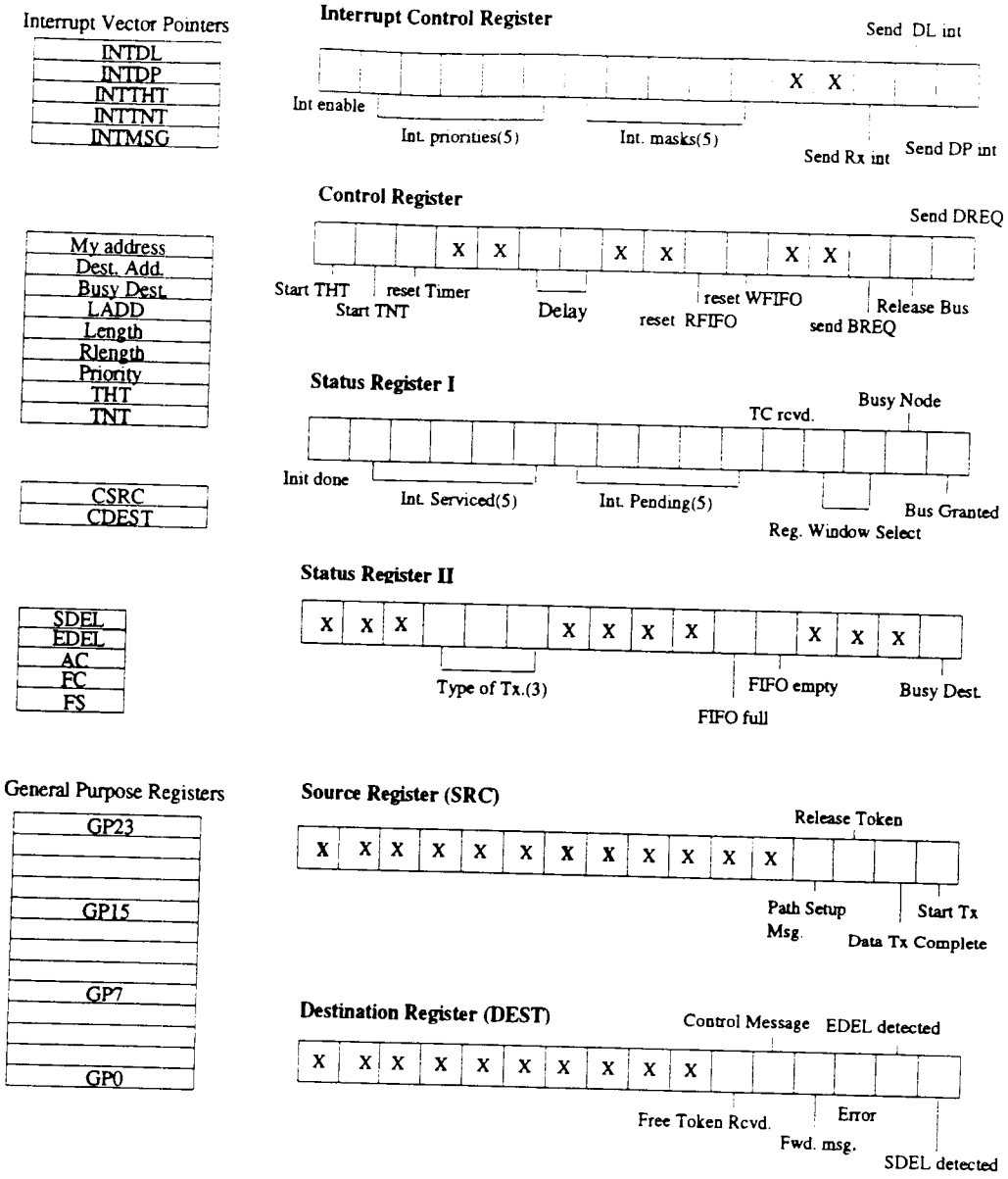


Figure 4.18 Programmable Registers for the Token Ring Interface.

The control frames like access control (AC), frame control (FC) etc. are programmable and depend on the type of transmission. The timer values for 'timer, hold token' (THT) and 'timer, no token' (TNT) depend on the length and number of nodes on the network along with the latency at each node and maximum message length on the ring. The timer values are programmable and should be selected such that the performance is optimized.

The data messages to be transmitted (received) over the ring are stored in the transmit (receive) buffer. The Tx and Rx buffers are 2K FIFOs which act as intermediate storage for messages to/from the host memory. It also provides temporary storage for control messages before being manipulated by the appropriate service routines.

Instruction format and instruction types are shown in Figure 4.19 and 4.20. They are similar to those presented for the data processor. The modifications are in the usage of some instructions and formats to accommodate the larger number of registers. Instructions have been provided to read and write from the FIFOs. The extra logic required to implement these instructions is not very complex.

Let us now take a detailed look at the receive and transmit sections of the token ring interface. Refer to Figure 4.21.

- The differential Manchester encoded packet arrives over the physical medium. The medium can be twisted pair or optical fibers.
- The data is decoded using the clock information extracted from the received packet.
- The delimiter decode logic continues to forward the message to the transmit section until it recognizes the starting delimiter (SDEL) flag.
- Once the SDEL is detected, the decode logic provides a path for the packet to the receive section.

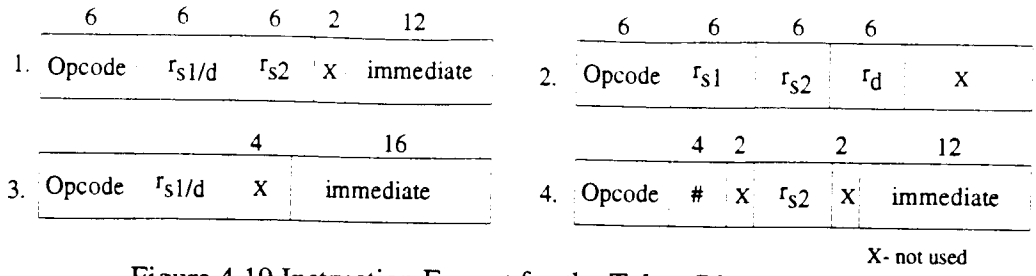


Figure 4.19 Instruction Format for the Token Ring Interface.

Instruction	Type	Function
LD reg. , [reg. + immediate]	1	Load the data from memory to the specified reg.
LDI reg.,[reg]	1	Load the data from memory and increment the memory add.
ST [reg. + immediate], reg.	1	Store the data from the reg. into the memory.
MV reg.,reg.	2	Move data from register to register
MVI reg., immediate	3	Move an immediate value to register.
ADD reg.,reg.,reg.	2	Add a reg. to reg. . Store the value in the third reg.
SBB reg.,reg.,reg.	2	Subtract a reg. from a reg. Store the value in third reg.
ADI reg., immediate	3	Add immediate to reg. Store the value in same reg.
SBI reg. immediate	3	Subtract immediate to reg. Store the value in same reg.
AND reg.,reg.,reg.	2	AND the contents of two regs. Store the value in third reg.
OR reg.,reg.,reg.	2	OR the contents of two regs. Store the value in third reg.
XOR reg.,reg.,reg.	2	XOR the contents of two regs. Store the value in third reg.
ANI reg., immediate	3	AND the immediate with reg. Store the value in same reg.
ORI reg., immediate	3	OR the immediate with reg. Store the value in same reg.
XORI reg., immediate	3	XOR the immediate with reg. Store the value in same reg.
RFIFO reg.	3	Read from FIFO into the given reg.
WFIFO reg.	3	Write to FIFO the contents of given reg.
BEQZ reg.,immediate	3	Branch if the value of reg. = 0.
BNEZ reg.,immediate	3	Branch if the value of reg. <> 0
STEQ reg.,reg.,reg.	2	Set the reg1 =1 if reg2= reg3
BSET reg.,#,immediate	4	Branch if the #bit of the register is set.
BRSET reg.,#,immediate	4	Branch if the #bit of the register is reset.
SHL reg.,reg.,immediate	3	Shift left the contents of reg. (No. of shifts = #immediate)
SHR reg.,reg.,immediate	3	Shift right the contents of reg. (No. of shifts = #immediate)
JMP immediate	X	Jump to the given address.
CALL immediate	X	Call the routine.
RET	X	Return

Figure 4.20 Instruction Set for the Token Ring Interface.

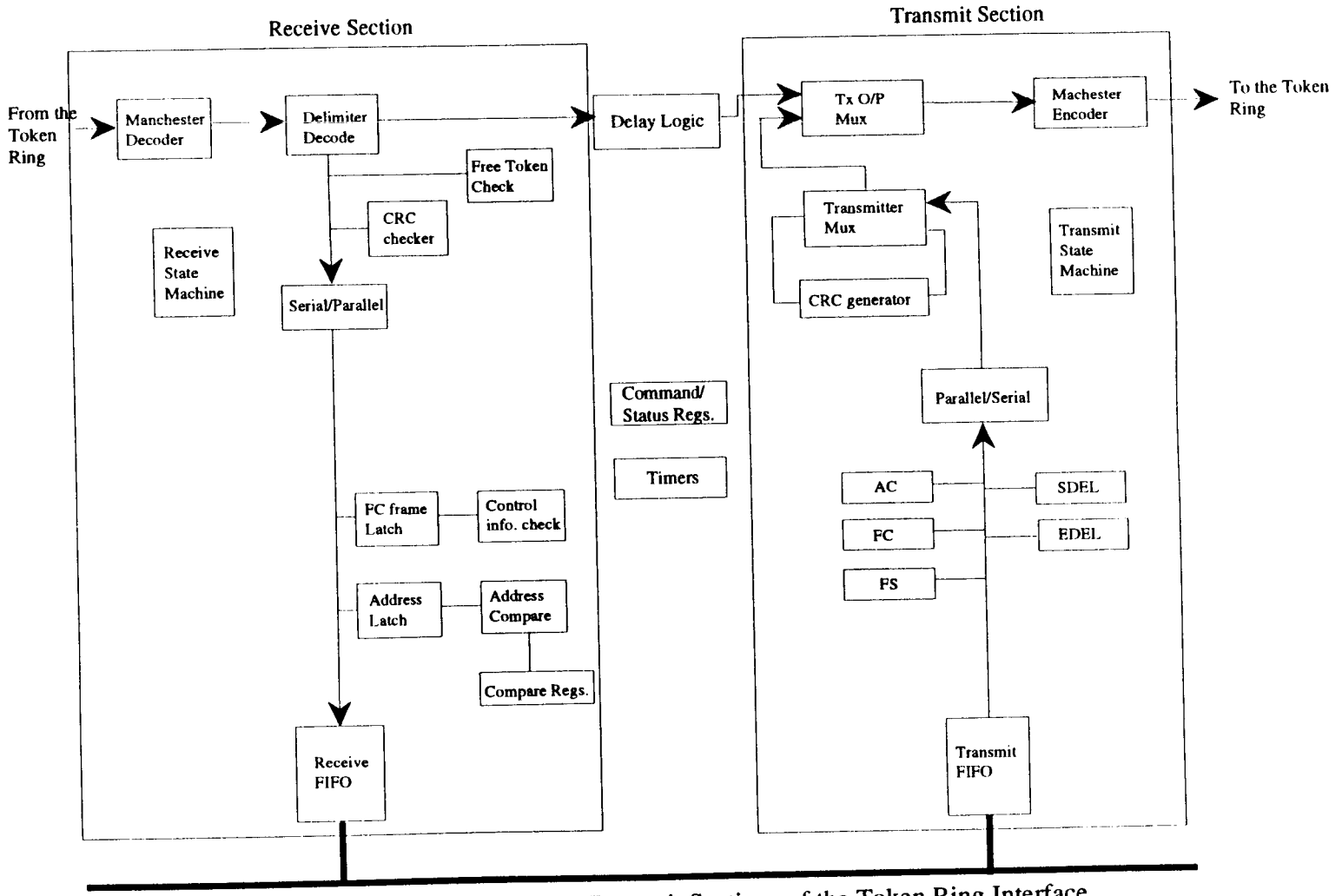


Figure 4.21 Receive and Transmit Sections of the Token Ring Interface

- The access control frame is checked for the free token bit, in the event of node's requirement to transmit. This is indicated by a 'start Tx' flag in the SRC register.
- Serial to parallel conversion of the received data occurs.
- The frame control (FC) is latched and checked for the type of packet (control or data). This sets an appropriate flag in the DEST register. In case of data packet, the destination address(DA) frame of the received packet is compared with the addresses in the compare registers.

  All the compare operations are time critical and are carried out in hardware under the control of receive state machine.
- The remaining packet are copied into the Rx FIFO under one of the following two conditions.

  1. If the control bit in the FC frame is set, the packet is copied irrespective of the destination address.

  2. If the packet is data, further frames are stored in FIFO only if the address match occured.
- The received packet is checked for errors. The CRC checker will generate the CRC for the received message and compare it with the one received. The error is indicated in case of a non-match. The type of CRC algorithm used is programmable but is uniform over the whole hybrid meshnet. The IEEE 802.5 standard uses a generator polynomial of degree 32 as shown here.

  $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + 1$
- A node wishing to transmit sets the 'start Tx' flag in the SRC register. Once the receive section detects the token, the TRI is switched to transmit mode, disabling the forwarding of data from the receive to the transmit section.

- The control frames at the Tx section are programmable. They are programmed with appropriate values before the transmission initiation.
- The data to be transmitted is stored in Tx FIFO. The data from the host to the FIFO is transferred under DMA control.
- The transmission of data involves parallel-to-serial conversion, CRC generation and the Manchester encoding.

Transmit and receive sections are similar to the one implemented for IEEE 802.5 standard token ring interface. Modifications are mainly required to accept control messages without checking destination address and provide capability to initiate transmission of control packet in the absence of free token.

Implementation of sample protocol routines allowed us to determine the register and memory requirements. It also enabled us to make timing and performance measurements which are discussed in the next chapter. Token ring functions are carried out through a set of interrupt routines.

- Token ring interface has various programmable parameters. The host system initializes the processor by sending a PG2 interrupt (refer to Figure 4.1).
- On interrupt from the data processor, the control information from the shared memory is received. The decision based on the information results in the initiation of control or data transmission. The control frames are generated accordingly. The data transmission involves transfer of data from the host memory to the Tx FIFO.
- On interrupt due to the non-empty Rx FIFO, the routine examines the control flag in the DEST register. If the received packet is data, it transfers the data to the host via DMA. In case of a control packet, it determines the action and accordingly sends message to the data link interface and/or the data processor.

- On interrupt from the data link interface, it send either the path grant or the path release packet over the ring.

The token ring functionalities are implemented using the suggested instruction set. This is shown in Appendix A. The sample implementation is approximately 550 instructions long. Though it is not complete, it allows us to make an estimate on the overall memory requirements. Based on this, it can be estimated that the instruction memory of size 4-8 Kbytes will be sufficient for the token ring protocol implementation.

4.5 Data Link Interface

The data link interface is responsible for handling the transfer of messages over the data channels. It implements all the MAC layer and few LLC layer functionalities associated with this transfer.

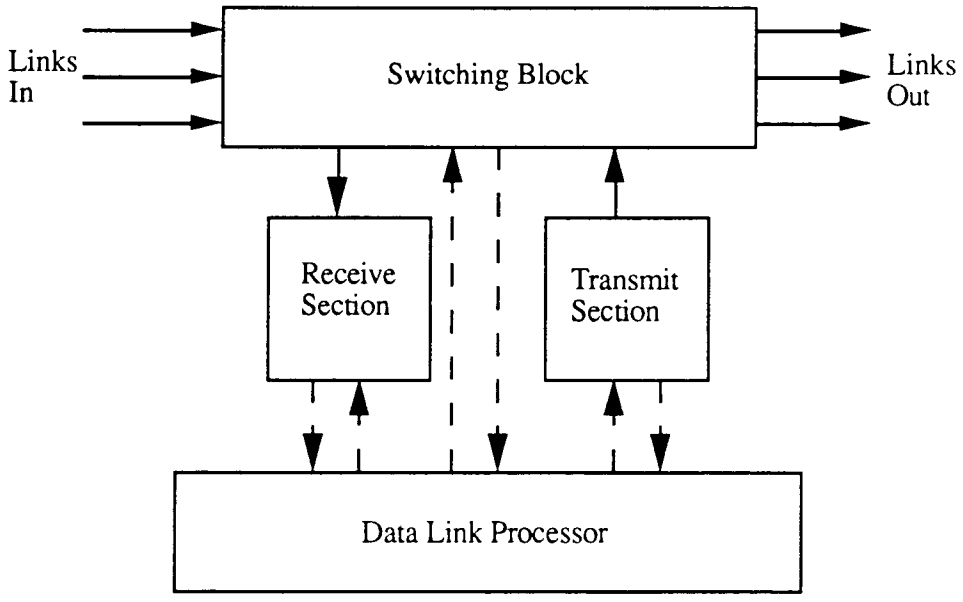


Figure 4.22 Block Diagram for the Data Link Interface

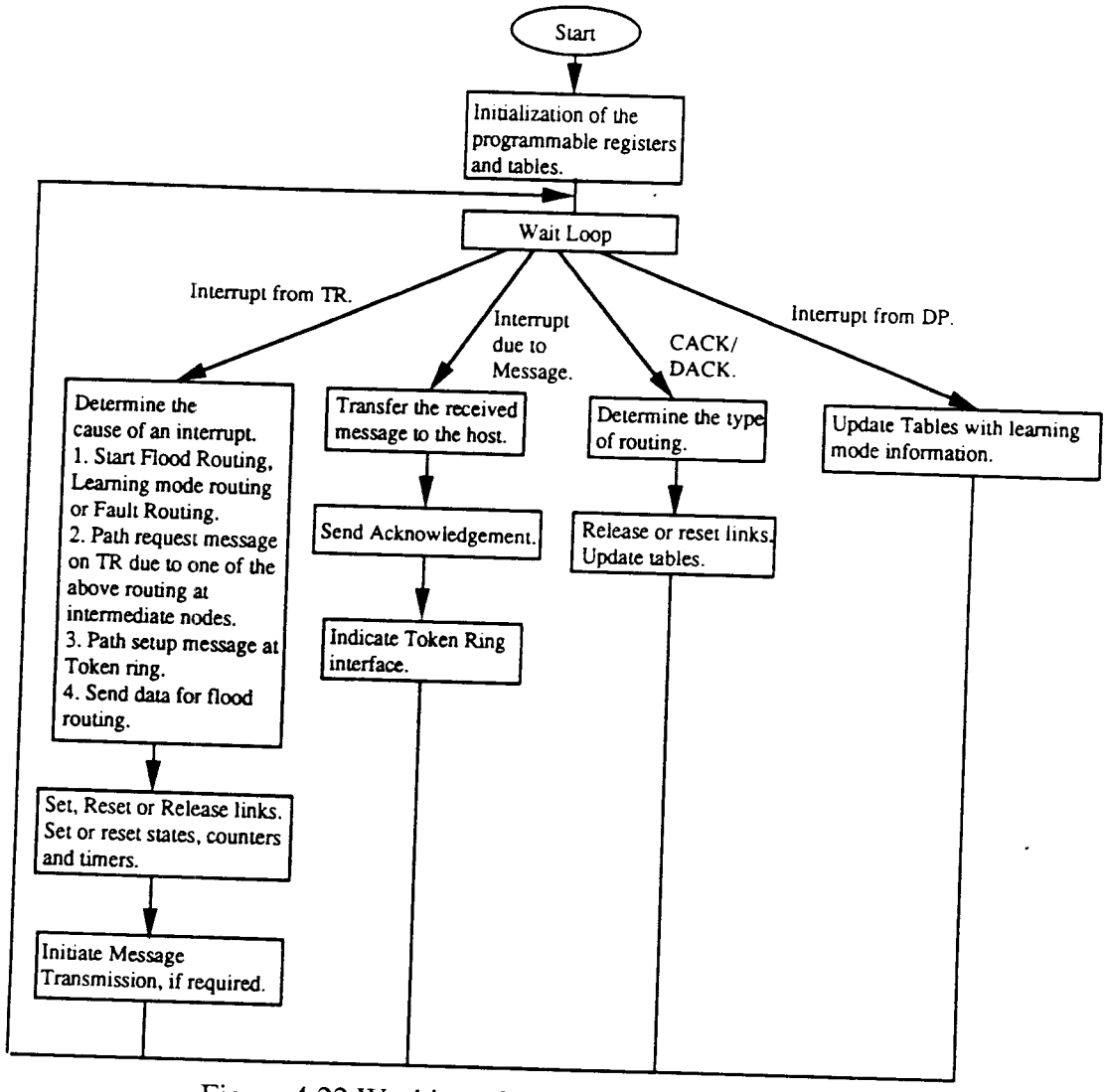


Figure 4.22 Working of the Data Link Interface.

At each node, the data link interface (DLI) consists of the switching blocks, the receive section, the transmit section and the data link processor. The working of the data link interface is shown in Figure 4.23. The basic block diagram is as shown in Figure 4.22.

The data processor architecture has the same basic architecture as described in an earlier section. The instruction format and the types are as shown in Figure 4.24 and 4.25.

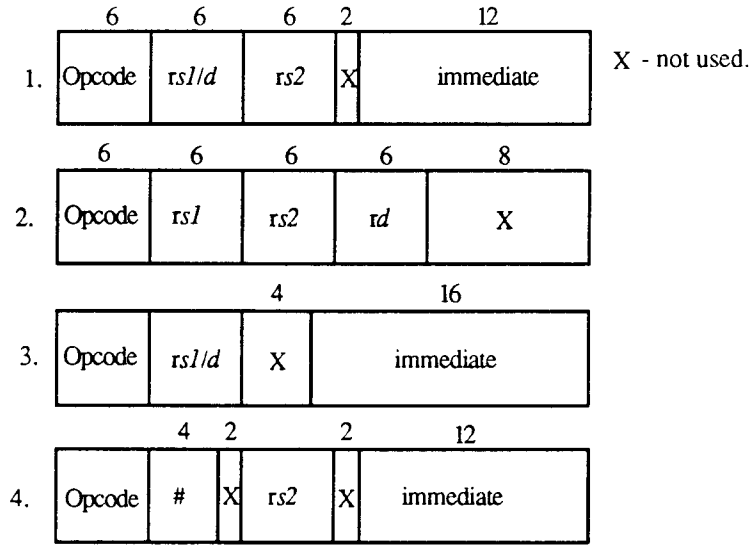


Figure 4.24 Instruction Format for the Data Link Interface.

New instructions have been added to implement special functions. To accommodate these instructions and allow their execution to be at the rate of one per cycle, additional functional blocks are incorporated into the basic architecture. The preliminary logic design of the functional blocks indicate that the required changes are not very complex. The addition of instructions and logic is justified on the following grounds.

1. The specialized functions can be carried out in 1-3 cycles which could otherwise require many cycles to accomplish.

Instruction	Type	Function
LD reg. , [reg. + immediate]	1	Load the data from memory to the specified reg.
LDI reg.,[reg]	1	Load the data from memory and increment the memory add.
ST [reg. + immediate], reg.	1	Store the data from the reg. into the memory.
RFIFO reg.	3	Read from FIFO into the given reg.
WFIFO reg.	3	Write to FIFO the contents of given reg.
MV reg.,reg.	2	Move data from register to register
MVI reg., immediate	3	Move an immediate value to register.
ADD reg.,reg.,reg.	2	Add a reg. to reg. . Store the value in the third reg.
SBB reg.,reg.,reg.	2	Subtract a reg. from a reg. Store the value in third reg.
ADI reg., immediate	3	Add immediate to reg. Store the value in same reg.
SBI reg. immediate	3	Subtract immediate to reg. Store the value in same reg.
AND reg.,reg.,reg.	2	AND the contents of two regs. Store the value in third reg.
OR reg.,reg.,reg.	2	OR the contents of two regs. Store the value in third reg.
XOR reg.,reg.,reg.	2	XOR the contents of two regs. Store the value in third reg.
ANI reg., immediate	3	AND the immediate with reg. Store the value in same reg.
ORI reg., immediate	3	OR the immediate with reg. Store the value in same reg.
XORI reg., immediate	3	XOR the immediate with reg. Store the value in same reg.
SHL reg.,reg.,immediate	3	Shift left the contents of reg. (No. of shifts = #immediate)
SHR reg.,reg.,immediate	3	Shift right the contents of reg. (No. of shifts = #immediate)
BEQZ reg.,immediate	3	Branch if the value of reg. = 0.
BNEZ reg.,immediate	3	Branch if the value of reg. <> 0
STGE reg.,reg.,reg.	2	Set the reg1 =1 if reg2 >= reg3
STLE reg.,reg.,reg.	2	Set the reg1 =1 if reg2 <= reg3
STEQ reg.,reg.,reg.	2	Set the reg1 =1 if reg2 = reg3
BSET reg.,#,immediate	4	Branch if the #bit of the register is set.
BRSET reg.,#,immediate	4	Branch if the #bit of the register is reset.
ENC reg.	3	Encode(16->4) the value in ENCODE and store it in reg.
DEC reg.	3	Decode(4->16)the value in DECODE and store it in reg.
STLN	X	Set the available data links in the switch reg.
SSTLN	X	Set the available data links in the source switch reg.
RLSLN	X	Release the unused data links in the switch reg.
SRLSLN	X	Release the unused data links in the source switch reg.
RSTLN	X	Reset the used data links in the switch reg.
SRSTLN	X	Reset the used data links in the source switch reg.
LDSWR @R0	3	Load Switch registers specified by R0.
STRST @R0.,immediate	3	Store the state in registers specified by R0.
RSST @R0	3	Reset counter.
RSSQ @RO	3	Reset switch registers.
JMP immediate	X	Jump to the given address.
CALL immediate	X	Call the routine.
RET	X	Return

Figure 4.25 Instruction Set for the Data Link Interface.

Consider the instruction 'DEC reg.'. It converts a 4-bit representation of a number to a 16-bit representation. This can be implemented using the basic instruction set (shifts, additions/subtractions and branches) at the expense of many clock cycles. A decoder and a special register (DECODE) can carry out the same function in two clock cycles.

2. The new instructions are designed to limit the number of registers accessed directly to below 64.

There can be up to 16 links supported by the data link interface. Each one will have its own switch and state registers (explained later in the section). Trying to access each register independently will require the instruction format to support more than 64 registers. The number of bits in the instruction will be more than 6 for each register field. The instruction length will be more than 1 word (32 bits) long. This is unacceptable. By providing a way of indirect addressing, the registers required to be referenced directly are kept below 64.

Consider the instruction STRST @R0,immediate. The instruction stores the immediate value into each of the state registers specified by R0. Each bit in R0 represents a link. The register at a link having the corresponding bit set in R0 is loaded with the immediate value. This allows for 16 registers being specified by a single instruction and a single register.

The instruction format differs from the one presented for the data processor. This is to accommodate a larger number of registers. The instruction set has many special purpose instructions.

The register organization of the data link interface shown in Figure 4.26. represents the register array shown in Figure 4.6. The data link interface can receive external interrupts from the data processor and the token ring interface. Internally the interrupts are generated due to a reception of an acknowledgement or a data message.

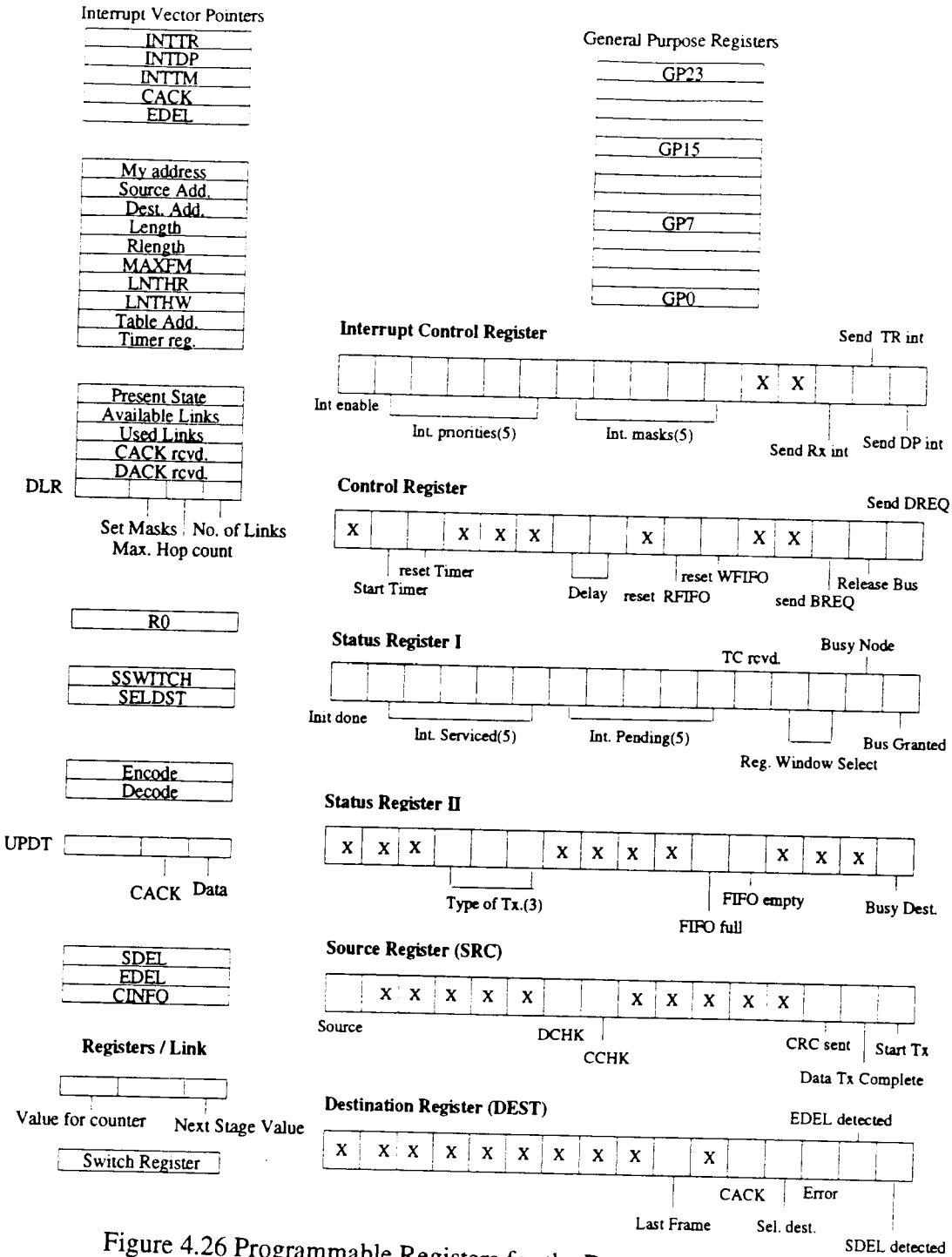


Figure 4.26 Programmable Registers for the Data Link Interface

Interrupt vector pointers provide the addresses for the corresponding interrupt routines. These addresses are programmable at initialization. Functions of the control and status registers are similar to those of the token ring interface. SRC and DEST registers provide control and status information for the transmit and receive sections of the interface.

Registers are provided to keep track of the size of data received (LNTHW) and data read (LNTHR). The data link interface, based on the information learned during the learning mode routing, maintains a table in the shared memory. The table provides an immediate node address for each of the outgoing data links. The 'Table Add.' register points to the beginning of this table and is accessed during fault routing.

The present state, the available links and the used links registers indicate the current state of the switching block. Each bit in these registers correspond to a link. A 0 in any position indicates a free link. These three registers are required to manipulate the switching during the different stages of transmission. ENCODE and DECODE are the special registers used by the ENC and DEC instructions respectively.

The transmit and receive sections at the source and destination nodes are associated with sending message over the data links. They are shown in Figure 4.27a and 4.27b. These sections are similar to those of the token ring interface. Each section is provided with 4K FIFO. Under the present implementation, once a path is setup between the source and the destination node, a maximum of 64 Kbytes can be transferred with maximum packet size being 4 Kbytes.

The switching block is shown in Figure 4.28. It performs various functions. It can recognize data and control messages, switch stages, verify data integrity, remove duplicate messages, check for hop count, append source address to the message, switch to the available links and release unwanted links. It is required that all these functionalities are carried out with minimum delay in transmission.

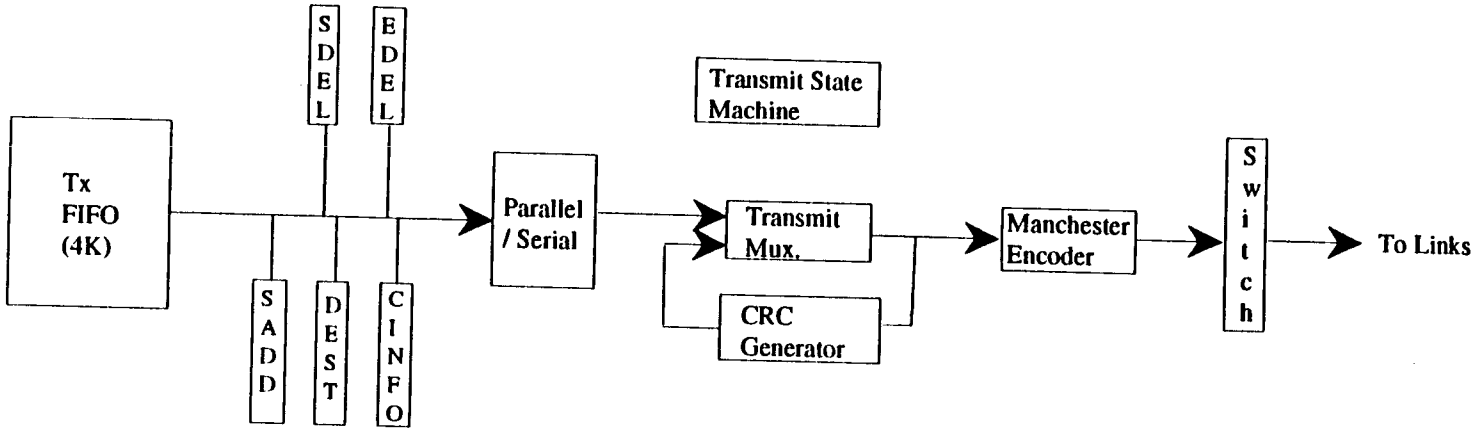


Figure 4.27a Transmit Section of the Data Link Interface

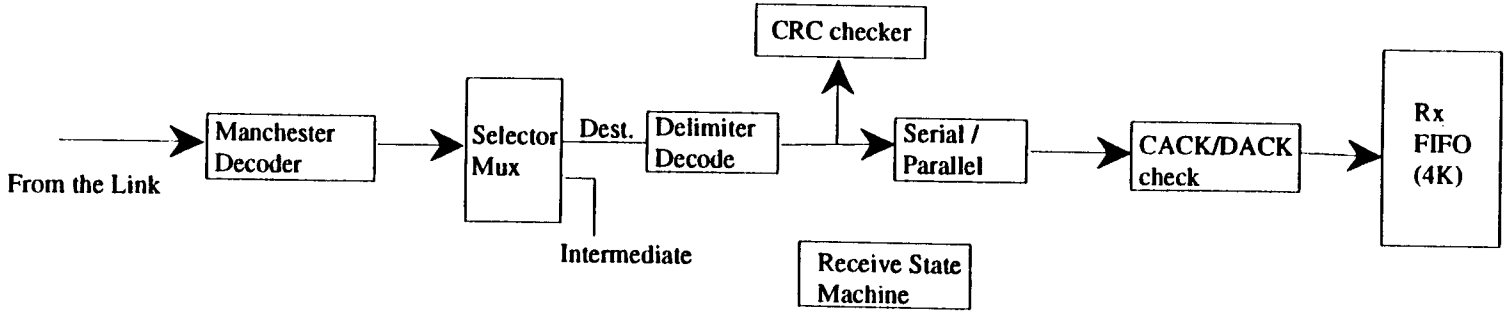


Figure 4.27b Receive Section of the Data Link Interface

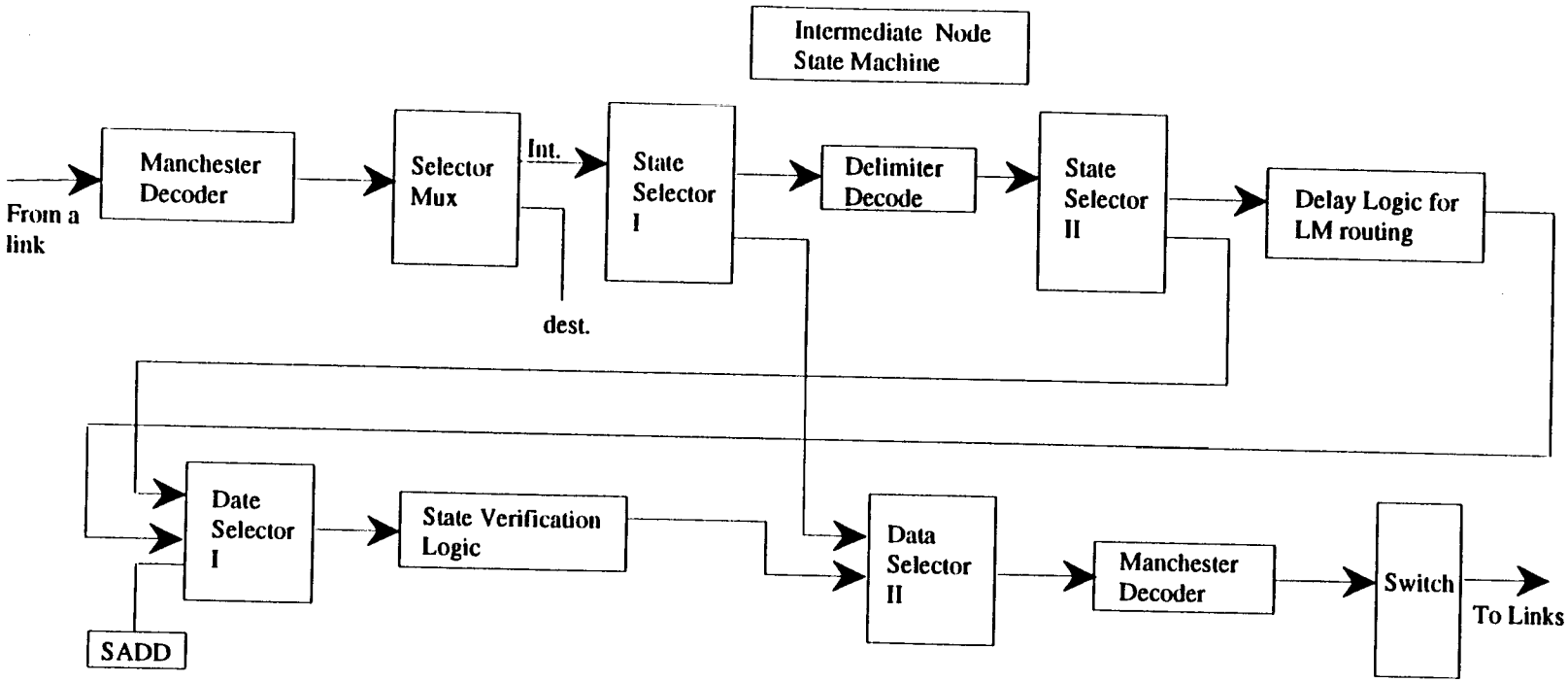


Figure 4.28 Switching Block for the Data Link Interface.

Each link has its own switching block. The switching block can be in any one of the following six states. If the frame received does not conform to the state of the link, the frame is rejected and is not forwarded.

0. Idle
1. Receive path search message : The switching block at the intermediate node checks for the hop count in the control frame. The frame is discarded if the hop count is greater than the maximum allowable value. Under the present implementation, the maximum possible hop count is 16. All the available links are set to receive the path search message. When one of the links receives the message, all the other links are switched to state 2. The link receiving the message will be switched to state 3. If the node receives another search message, it will be discarded preventing duplicate messages. This state introduces a 5 bit delay at each node in the path.
2. Receive CACK : The switching block checks for the CACK bit in the control frame. On recognition, it releases all the unused links. The link is switched to state 4 or state 0 depending on the type of routing. A 1 bit delay is provided to check the control bit.
3. Forward Data Message : Since the path is already setup, the data is forwarded at the intermediate nodes without any intervention or delay.
4. Receive DACK : The switching block checks for the DACK bit in the control frame. On recognition , it resets the links forming the path. The links are switched to state 0. A 1 bit delay is provided to check the control bit.
5. Path search for Learning mode routing : This is similar to state 1. The difference being that the node address is inserted in the path search message by the switching block. The delay is 21 bits at each node in the path.

The message inside the switching block is routed, depending on the state of the link, to carry out the functions associated with the state. Switching of the links, finding available links, setting up links, releasing unused links, resetting used links and switching of the states are done under a program control and are dependent on the state of the link.

Intermediate state machine consists of a counter and few registers to generate the next state and enable the proper checking of various control bits. Multiplexers and Selectors have outputs of the programmable registers as the control inputs.

Switches are of the crossbar type with the switching block at the input of each link as shown in Figure 4.29. By implementing the various stages of the switching block at the logic level, it was realized that its design may not be trivial. However, the design remains the same for all the links. Multiple switching blocks can be obtained by duplication.

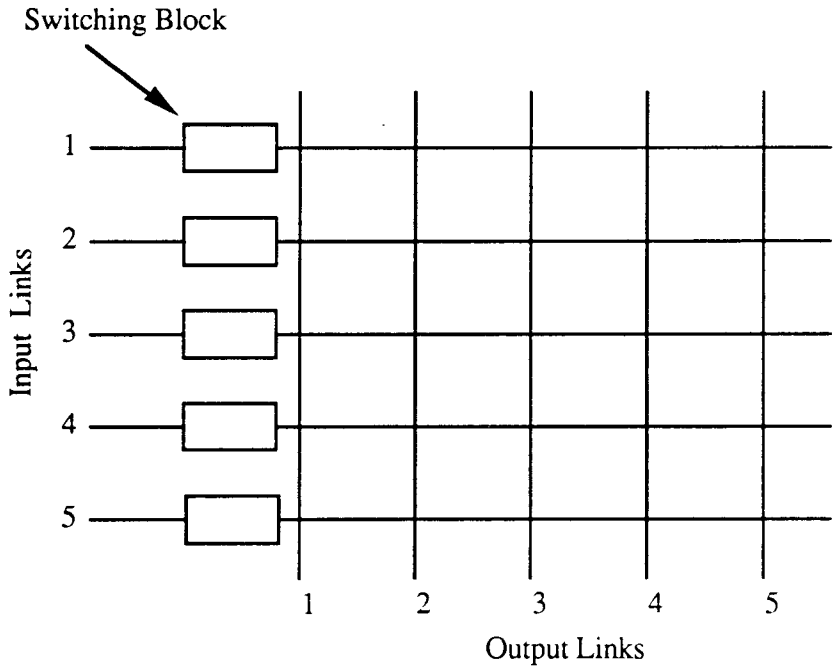


Figure 4.29 Switching Section of the Data Link Interface.

The data link interface functions are carried out through a set of interrupt calls.

- Data link interface has various programmable parameters. The host system initializes the processor by sending a PG3 interrupt (refer to Figure 4.1).
- On interrupt from token ring, the control information is read. Setting and resetting of the links, writing into the state registers and initiation of transmission is done as per the control information.
- On reception of the data message, an interrupt occurs at the destination. The data from the Rx FIFO is transferred to the host memory using DMA.
- The data processor provides the information about the immediate node addresses connected to the links at each node. On interrupt from the data processor, the table pointing to these links is updated.

Sample protocol implementation for the data link interface is shown in Appendix A. The implementation is quite complex and requires a total understanding of the switching blocks in particular. The code is based on the instruction set shown in Figure 4.25 and is 650 instructions long.

For the current implementation, the data link interface supports 16 data links. This implies that any particular node can be connected to other 16 nodes via point-to-point links.

CHAPTER 5

PERFORMANCE EVALUATION

The design features for the network interface unit were presented. An evaluation of the designed unit will be discussed in this chapter. This exercise will enable us to determine several pieces of valuable information:

- Determine and understand the advantages of flood routing over the broadcast transmission.
- Estimates for the timing requirements for learning mode routing and flood routing can be made.
- Programmable parameters like broadcast data transmission reference number (BDRN), timer no token (TNT) and timer hold token (THT) need to be determined. It is possible to estimate these values based on the sample implementation of the protocol and various network parameters.
- Processor utilization is studied and the possible improvements are suggested.

To carry out the above mentioned objectives, it is required to know :

1. *The time required by the processors to carry out the required functions* (path request, path setup, send data etc.). - The sample implementation of the protocol allows us to determine this time. Knowing the instruction count for each of the routines, processor speed, waiting time for for free token, transmission time etc., it is possible to estimate the total processing time.
2. *Processor speed* .- Each processor is assumed to work at 20Mhz (50ns. clock cycle).
3. *Availability of the shared bus for the unit.*- The units of the NIU share a common bus for transfer of status informations. If the bus is free, it is given to the requesting processor at the next cycle (50ns). Maximum wait corresponds to the time when the routing information is manipulated. This can take up to 16 cycles

(0.8 μs). For our analysis, we assume the average wait time for the bus to be 0.4 μs.

4. *Transmission rate on token ring and data channels* - The IEEE 802.5 standard for the token ring specifies the transmission rate to be either 4 Mbps or 10 Mbps. We will use both of these values. The transmission rate for data channels is assumed to 10 Mbps.
5. *Time to wait for free token* - The time before which a free token is available depends on number of parameters: the network load, the number of nodes in the network, the average length of messages transmitted etc. The minimum token wait can be 1 bit which is equivalent to 0.25μs (4 Mbps) or 0.1μs (10 Mbps). The source has to wait the longest when all the nodes in the ring, beginning with the one closest to the source, would like to transmit. This time can be anywhere up to 1 sec. For our evaluation we will vary the token wait time value from 0.1μs to 100 ms.
6. *Propagation Delay* - This is assumed to be 5μs/km.
7. *Delay (latency) at each node*- The latency at each node in the case of token ring is 1 bit/node. In case of transmission over the data links the delay is dependent on the state of transmission. During the path setup stage, the latency is 5 bits/node. After the setup, the data transmission take place without any delay at the intermediate nodes.
8. *Number of nodes on the network* - The analysis is done for values of 25 and 100.
9. *Number of nodes in the path* - This is significant for transmission over the data links. For the analysis the number is varied from 1 (minimum) to 16 (maximum).
10. *Data and control message lengths* - The control messages sent over the token ring are 12 bytes long. The control messages on the data links are 11 bytes. The data length for the token ring is varied from 15 bytes to 6000 bytes and for the data channels from 100 bytes to 64 K bytes.

11. *Length of the token ring* - It is assumed to be 1 km.

It should be emphasized here that our analysis does not involve network traffic issues. No attempt has been made to determine the throughput of the hybrid meshnet, its operation under various load conditions and different arrival and service rates of the messages. This study has already been done in [1]. Our purpose of analysis is to determine the time required to transmit the data messages over the token ring and the data links and provide the comparisons between them. The queuing models are not considered for the analysis. The message lengths and the waiting time for token are varied over a range of values.

5.1 Results

Routines were written in C to determine the setup and transmission times for different routing mechanisms. The results are shown here in the form of graphs. Let us understand each of these graphs.

1. Figure 5.1 shows the message transfer time for different message lengths in case of broadcast transmission. This is shown for two different transmission rates. The message transfer time depends on the processing time at the NIU, waiting time for a token, the availability of the shared bus, the propagation delay and the transmission time. All the values except the transmission time (due to variable length of message) form a constant portion of the equation. As seen the message transfer time increases linearly with the message length. A particular value of interest is the time required for transmission of control messages (L=12bytes). It is 62.1 μs for 4Mbps and 44μs for 10Mbps.
2. Figure 5.2 shows the setup time as a function of waiting time for a free token for the flood routing. The setup time depends on the same factors as the broadcast transmission. However for this analysis, waiting time for the token, the

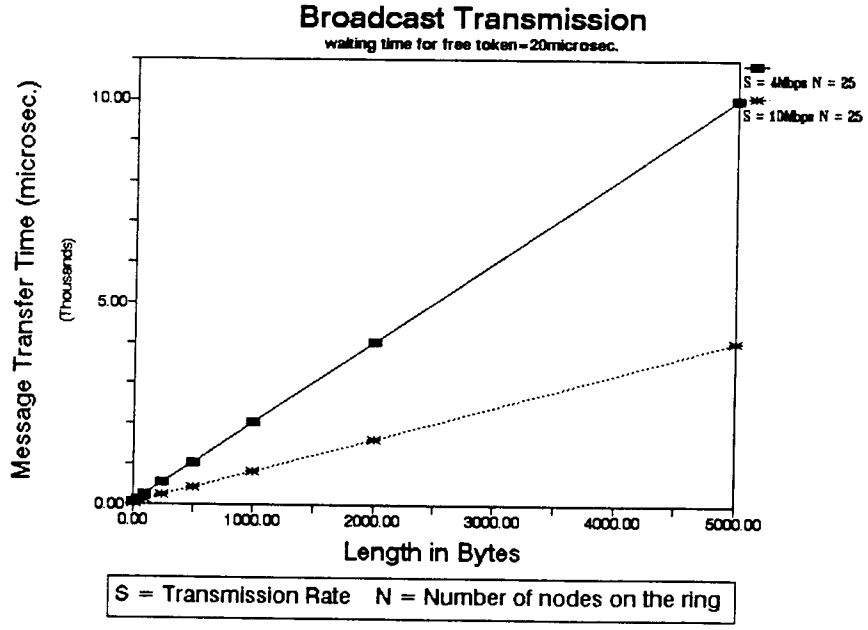


Figure 5.1 Message Transfer Time for Broadcast Transmission.

propagation delay (because of different values of N, number of nodes on the ring) and the transmission time (because of different transmission rates) are the variables. Since the flood routing and broadcast transmission are compared for a free token wait time of 20μs, it is of interest to find the setup times for that value. They are :

- 110.5μs for S = 4 Mbps and N= 25.
- 140.1μs for S = 4 Mbps and N= 100.
- 77.6μs for S = 10 Mbps and N= 25.
- 88.9μs for S = 10 Mbps and N= 100.

3. Figure 5.3 allows us to compare the setup times and message transfer times for a range of message lengths. As seen, the setup time is independent of the length of the message. It can be noted that the time required for path setup is greater than the time required to send messages, for smaller message lengths. For L = 100

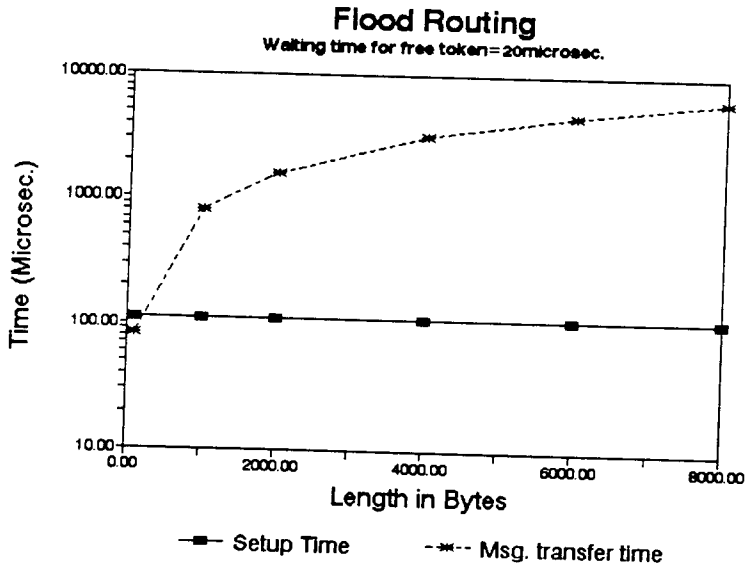


Figure 5.2 Setup Time in Flood Routing.

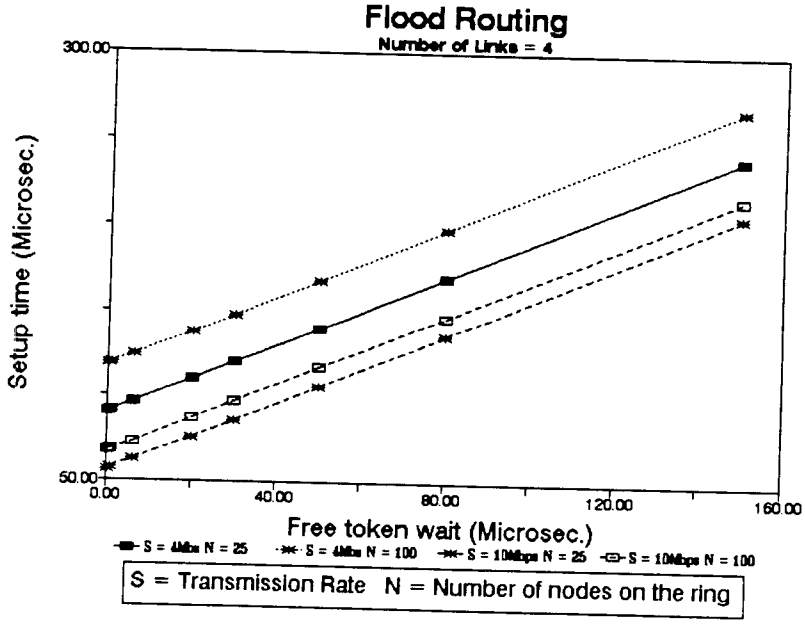


Figure 5.3 Setup Time and Message Transfer Time for Flood Routing

bytes, the setup time is 110.5 μs and the message transfer time is 82.9μs. This is an important criterion in determining BDRN. This will be discussed later in the section.

4. Figures 5.4 and 5.5 show the comparison between timings for broadcast transmission and flood routing for various message lengths. If the transmission rate on the ring is 4 Mbps, it is seen that the flood routing is always a better choice. However, as seen in Figure 5.5, the broadcast transmission takes less time than the flood routing for any message length if the transmission rate is the same on both the media. The reason being the setup time required for flood routing.

   The advantages of multiple transmissions due to the presence of the data channels can be appreciated from this graph along with Figure 5.3. Let us understand this numerically:

   The flood routing uses the token ring only during the setup time. Once the path is setup the token ring is free for use by the other nodes. This setup time is 88.9μs. Suppose we have two nodes. Each is required to transmit messages of length equal to 2000 bytes. Message transfer time in case of token ring (T_{ring}) is 1641.9 μs /message. The transmission of two messages will require 2*1641.9 μs. The message transfer time (T_{link}) in case of data channels (assuming that the path exists between the nodes) is 1591.1μs. In order to transmit two messages it will require 2*88.9 + 1591.1 μs.

   Total saving in time = Total transfer time over the ring - Total transfer time over the links.

   = (2*1641.9) - (2*88.9 + 1591.9)

   = 1514.1μs.

   The advantage becomes more apparent for longer message lengths and more concurrent transmissions. Using data channels for transmission also prevents the

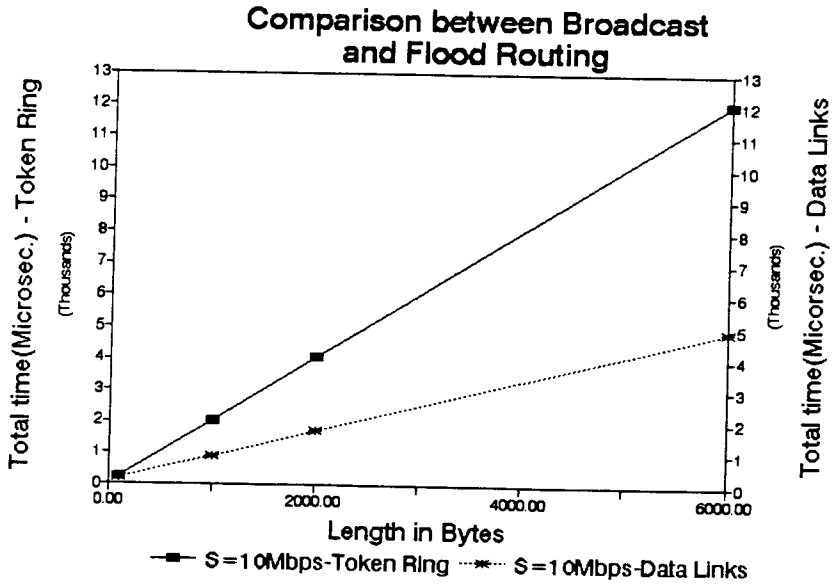


Figure 5.4 Comparison between Broadcast Transmission and Flood Routing.

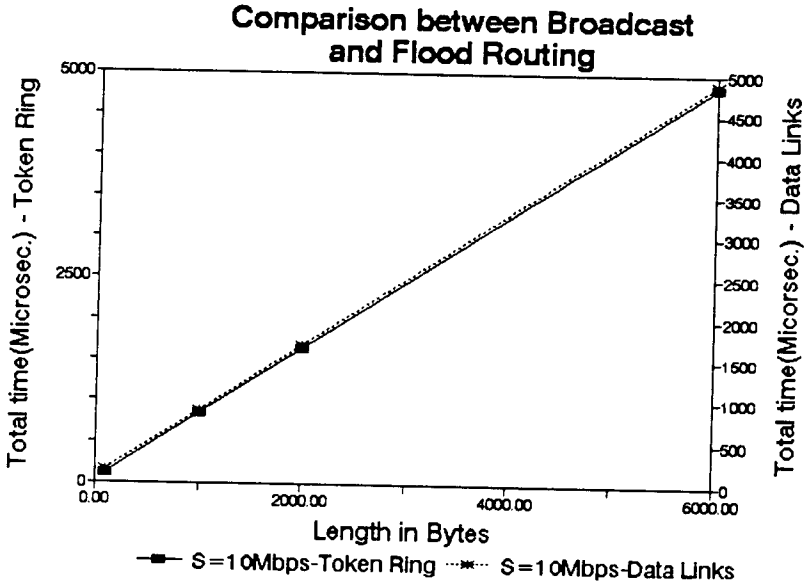


Figure 5.5 Comparison between Broadcast Transmission and Flood Routing.

token ring being clogged by longer messages. This indicates that the token ring should be used for control messages and short data messages while longer messages can be transmitted on the data channels.

5. Figures 5.6 and 5.7 show the timing requirements for the learning mode and the fault routing.

5.2 Determination of BDRN, THT and TNT

The BDRN value is used to determine the messages to be transmitted on the ring. The value can be the length in bytes for which the broadcast transmission time equals the setup time required for flood routing. However, the following analysis is based on the assumption that more than one node in the network desires to transmit long messages at any give time. From the above discussion, it can be seen that it will be efficient to send messages over the data channels. However, the transmission on data channels requires a setup time. If the message transmission time is less than the setup time, flood routing will be inefficient and will require more time than the token ring (assuming bit rate = 10 Mbps for token ring). It is found that for L = 100 bytes, both these parameters are almost equal. The BDRN value can be selected such that 2*setup time + T_{link} < 2*T_{ring} (assuming only two concurrent transmissions). From the results obtained during the analysis it is found that this will occur at message length (Lappx) > 200 bytes. Therefore BDRN = 200. One important point to note here is that this value is programmable at each node and can be different at different nodes depending on the needs of each node. More extensive analysis is required to determine the precise BDRN value.

THT, timer hold token defines the maximum time for which a node can hold the token. In order to determine this value, it is sufficient to find the longest message that

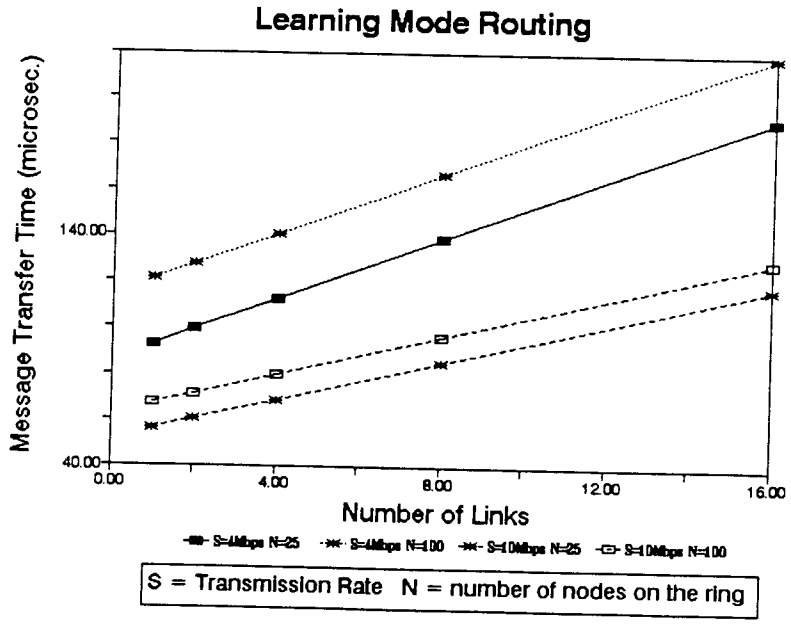


Figure 5.6 Message Transfer Time for Learning Mode Routing.

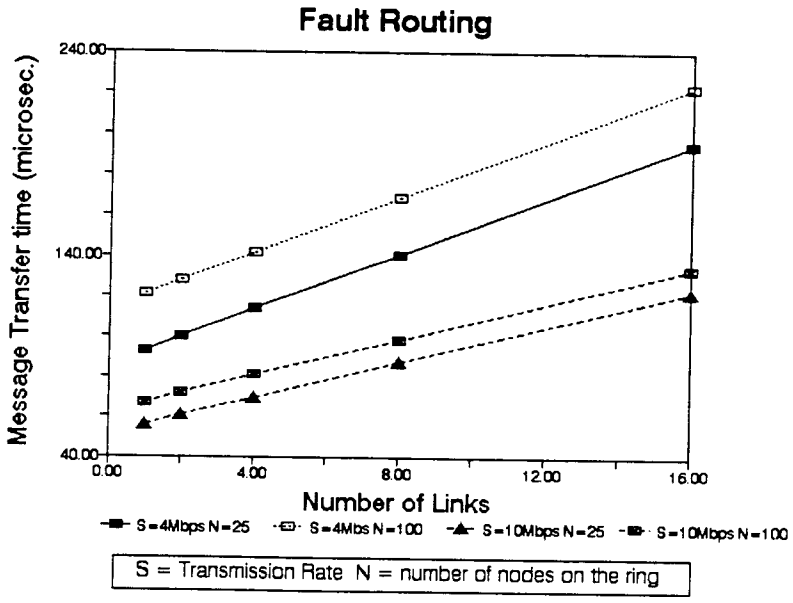


Figure 5.7 Message Transfer Time for Fault Routing.

will be transmitted on the token ring. This will be equal to the BDRN value. For BDRN = 200 bytes, the time required to transmit is 242μs. We can select THT = 250μs.

TNT, timer no token is defined as Number of nodes*THT. Assuming the number of nodes on the network to be 100, TNT = 25ms.

5.3 Processor Utilization

It is found that the time required for processing is negligible compared to the time required for the transmission. An approximate measurement indicates that only 2-2.5% of time is used in processing by a unit of the NIU. This indicates that the NIU is capable of supporting much higher transmission rates. In the sample implementation, the processor waits in a loop until the message is fully transmitted or received. Hence it is busy during the full transmission time. This is inefficient. It can be modified so that the processor is freed once the transmission is initiated.

The NOPs (No operation instructions) are included to avoid the control hazards in the RISC architecture. In the sample implementation, it takes as much as 15% of the time and code. The code should be reorganized such that more branch-delayed slots are filled with useful instructions instead of NOPs. This will further improve the processing time and the efficiency.

CHAPTER 6
RESULTS AND CONCLUSIONS

Previous chapters of the study considered the design features necessary to implement the network interface unit for the hybrid meshnet local area network. Hybrid meshnet protocol is still in its development stage. One of the design goals was to provide a unit which will support this ongoing changes. Except for the time critical and non-varying functions, all the functionalities of the NIU are programmable. This will provide for tremendous flexibility in the design, validation and standardization of hybrid meshnet. The multiprocessor system for the network interface unit ensures high performance. All the processors work concurrently on the tasks assigned to them. Various asynchronous activities are distributed among the three processors resulting in a balanced network interface unit.

The processors have a RISC architecture with simple instruction sets. The resulting control units for the processors will be simple. As the basic architecture remains the same for all the processors, the initial design can be duplicated for all the three processors.

The network interface unit will be compatible with most of the host computer systems. Only the host system interface needs to be modified to manage the timing and control signal conversion between the host system and the NIU.

The sample implementation of protocols for each of the interfaces allowed us to determine the instruction and register requirements and verify the programmability aspect of the network interface unit. The processor utilization and overall performance estimations were done based on this implementation. This exercise formed an important tool in the overall design.

The performance and timing evaluation allowed us to understand and quantify the advantages of hybrid meshnet. Results showed that multiple, long message

transmission capability on data links can yield better transmission efficiency than token ring.

The NIU is designed for the computer network system with :

- up to 256 nodes on the network.
- up to 16 data links from each node.
- maximum of 16 nodes in the path between the source and the destination during flood routing.
- up to 64 Kbytes data messages.
- transmission rate of 10 Mbps on the data links and 4/10 Mbps on the token ring.
- modified IEEE 802.5 standard for token rings. In fact, the unit is designed such that in an absence of data link interface unit, the unit can work as a network interface unit for a token ring network.

The cost involved in developing the NIU may not make it competitive with the ethernet or the token ring for the 'day-to-day' networking environment. However, a feedback from the industry indicates that it definitely has a potential in a LAN environment requiring prompt, robust and flexible services. In the application environments, there are various kinds of data messages: large volume data transfer messages and a smaller volume of control signals. The hybrid meshnet finds use in this kind of application environments. The advantages of multiple, secure transmissions and reliability can be utilized in some applications like office automation, military C^3 (command, control and communication) and automated real time environments [1].

6.1 Areas of Further Study

The network interface unit presented here is a 3 processors system. The data processor makes decisions affecting the activities of the token ring interface and the data link interface. The designed system is modular and distributes tasks among the

processors. This results in a high performance system which is easy to design and easy to test. However if the cost is the main criterion, it may be possible to design the system with two processors. The functions of the data processor can be distributed among the token ring and data link interfaces. It may be worthwhile to discover the possible implications of this change.

The design features presented here brings us one step closer to the implementation. Suggested steps to reach this goal are:

- Behavorial simulation of the NIU.
- The attachment of the NIU with the physical media is not addressed in this study. It is necessary to address this issue during the early stages of implementation.
- Design of the basic RISC processor. The instruction set implemented at this stage can be the one which is common for all the processors.
- Design of the receive and the transmit sections for the token ring interface and data link interface can be done concurrently with the design of the processor. The transmit and receive sections for token ring can have some ideas borrowed from the ones available commercially.
- The functionalities of each interface is different from the others. Additional logic will be required to carry out these functions. They can be incorporated to the basic architecture for each of the interfaces.
- Protocol implementation can be done in assembly or an higher level language. The compiler design will be required for higher level language implementation.
- Depending on the host system , the host system interface needs to be designed.

Since the unit allows most of the protocol to be programmable, the protocol changes can be deferred until after the design. Some of the token ring functionalities like generating a new token in case of a lost token are not considered. However it should be possible to implement these functionalities with the existing register organizations and instruction sets.

The following points may be considered to improve the functionalities of hybrid meshnet.

Direct transmission : Direct transmission will allow message transfer between nodes connected by point-to-point links without sending control messages on the token ring. Feasibility of the direct transmission for the present system and the changes in protocol necessary for the implementation can be studied.

Fault diagnosis algorithms [1]: It may be possible to develop more efficient distributed fault diagnosis algorithms than the existing general algorithms. Fault recovery should also be considered.

Semi-permanent paths between the nodes: Allowing semi-permanent paths between the nodes can be very useful for real-time environments. The required changes in protocol should be studied.

BIBLIOGRAPHY

[1] Kang, Cheoul Sin. Hybrid Meshnet: A Local Area Mesh Network with Distributed Control. PhD. Dissertation, Oregon State University, 1987.

[2] Kang, C-S and Herzog J. H. Hybrid Meshnet: a new approach to mesh LANs. Proc. IEEE 13th Conference on Local Computer Networks, Oct. 1988, pp. 463-470.

[3] IEEE 802 Local Area Network Standard Draft. IEEE 802.5 standard 802.5 Token Ring Access Method and Physical Layer Specifications, 1989.

[4] Hutchison, David. Local Area Network Architecture. Addison Wesley, 1988.

[5] Byers, T. J. Microprocessor Communications and Support Chips. Elsevier, 1987.

[6] Henessy, John and Patterson, David. Computer Architecture and Quantitative Approach. Morgan Kaufmann, 1990.

[7] Stone, Harold. High-Performance Computer Architecture, 2nd Edition. Addison Wesley, 1990.

[8] Kang C-S; Park E. K. and Herzog J. H. Design of a Network Interface Adapter for Hybrid-LANs. IEEE 34th Midwest Symposium on Circuits and Systems, May 14-17, 1991.

[9] Krishnakumar A. S. and Sabnani. VLSI Implementation of Communication Protocols - A survey. IEEE Journal of Selected Areas in Communications, vol.7 no.7 Sept. 1989

[10] Hammond, Joseph and O'Reilly, Peter. Performance Analysis of Local Computer Networks. Addison Wesley, 1988.

[11] Stallings, William. A Tutorial on the IEEE 802 Local Network Standard. Computer Science Press, 1986. (pg 1-30).

[12] Ahuja, Vijay. Design and Analysis of Computer Communication Networks. McGraw-Hill, 1982.

[13] Keiser, Gerd. Local Area Networks. McGraw-Hill, 1989.

[14] Tanenbaum, A.S. Computer Networks, 2nd Edition. Prentice-Hall Inc., 1988

[15] Stallings, William. Local Netwoks, 3rd Edition. MacMillian Publishing, 1990.

[16] Advanced Memory Devices, Memory Product, 1989/1990 Data Book, 1989.

[17] TMS380 Adapter Chip Set User's Guide. Local Area Network Products, 1986. Texas Instruments Inc.

[18] Davis, Donald and Barbur, Derck. Communication Networks for Computers. John Wiley & Sons, 1976.

[19] Dixon, R,; Strole, N and Markov, J. A Token Ring Network for Local Data Communication. IBM Systems Journal, No.1, 1983.

[20] Bell Paul and Jabbour Kamal. Review of Point-to-Point Network Routing Algorithms. IEEE communications magazine, Jan. 1986 - vol.24, No.1, pp. 34-38.

[21] Chisvin Lawrence and Duckworth R.J. Content Addressable Memory and Associative Memory. IEEE Computer, July 1989, pp. 51-63.

[22] Hsieh Wen-Ning and Gitman Israel. Routing Strategies in Computer Networks. IEEE Computer, June 1984, pp. 46-56.

[23] Midkiff Scott and Caroll Christopher. Architectural Support for Interprocess Communication in Point-to-Point Multiprocessor Networks. Sixth Annual International Phoenix Conference on Computers and Communications, 1987, pp. 14-17.

APPENDIX

```
/* ------------------------------------------------------------------
                            DATA PROCESSOR
   ------------------------------------------------------------------ */

/* This is the sample implementation of the protocol functions carried out in
   the data processor. The code is not complete or error free. It is not
   executed on any particular processor.
   However it has helped in determining the register and instruction
   requirements. This exercise will also allow us to estimate the performance.
   The control hazards that occur in pipeline architecture are taken care of.
   (MOVED in comments indicates that an instruction is rescheduled to avoid
   control hazard.
*/

/* The code indicates the register required for each function. The exact bit
   position is postponed until final implementation. */

/* Initialization routine. Until the PG1 interrupt is received the program
   waits in a loop shown later... */
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        MVI GP1,#----           ; address for DMAC ( host system interface ).
        MVI GP0,#----           ; set address.
        MVI GP2,#----           ; set length.
        MVI GP3,#----           ; command word.
/* data hazard which may result due to usage within two cycles of write
   has been taken care of in the architecture */
        ST [GP1+#0],GP0         ; store them in the DMAC.
        ST [GP1+#1],GP2         ;
        ST [GP1+#2],GP3         ;
        ANI ICR,ICR,#----       ; release the bus.
        MVI TC,#----            ; set up the TC interrupt vector register.
        RET                     ;
        ORI ICR,ICR,#- --       ; send DREQ. (MOVED)

/*      At Terminal Count (TC) interrupt.......
        Determine the cause of TC interrupt. */

        BSET SR1,#-,tx          ; check for Tx service flag.
        ORI ICR,ICR,#----       ; send BREQ ..
        NOP                     ;
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        NOP                     ;
        NOP                     ;
        LDI INTTR,[GP0]         ; interrupt vector address for TRI.
        LDI INTDL,[GP0]         ; interrupt vector address for DLI.
        LDI INTTX,[GP0]         ; Interrupt vector address for the host.
        LDI SADD,[GP0]          ; my address.
        LDI PRI,[GP0]           ; specify priority level...
        LDI BDRN,[GP0]          ; and length.
        LDI CNT,[GP0]           ; count for L.M. and F.R.
        LDI NNO,[GP0]           ; number of nodes in the network.
        LDI TABLE,[GP0]         ; starting address for routing table.
        LDI TIC,[GP0]           ; timer count.
        MVI GP1,#----           ; CAM address for decison table(DT).
        MVI GP2,#----           ; CAM address for routing table(RT).
        MV GP3,NNO              ; Initialization for decision table.
        LDI GP4,[GP0]           ; host provides the addresses of nodes on the
                                ; network.
cont:   ST [GP1+#0],#----       ; command word for Decision Table.
        ST [GP2+#0],#----       ; command word for Routing Table.
        ST [GP1+#1],GP4         ; write the address.
        ST [GP2+#1],GP4         ;
        MV GP5,CNT              ; store the LM and FR count in the table at
        SHL GP5,GP5,#-          ; the specified location along with
        ORI GP5,GP5,#----       ; other information.
        ST [GP1+#2],GP5         ; initialization sequence
        SBI GP3,GP3,#1          ;
        MVI GP5,#0000h          ; addresses for routing table are not known.
        BNEZ GP3,cont           ;
        ST [GP2+#2],GP5         ; moved here ..
        LDI GP4,[GP0]           ; acts as delayed slot (reqd. for loop)
        ANI ICR,ICR,#----       ; release the bus.
        ANI SR1,SR1,#----       ; reset the PG1 interrupt service flag.
        ORI SR2,SR2,#----       ; initialization done.
        RET                     ;
        ORI ICR,ICR,#----       ; moved..enable interrupts.
tx:     BSET SR1,#-,tx          ; wait for the bus.
        NOP                     ;
        NOP                     ;
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        NOP                     ;
        ORI SR1,SR1,#----       ; set the resgister window.
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,SR1,#----       ; set the other window.
ahead:  LDI GP1,[GP0]           ; control information.
        LDI GP3,[GP0]           ; priority
        LDI GP4,[GP0]           ; BDRN
        STGE GP3,GP3,PRI        ; depending on the priority and BDRN make
        BNEZ GP3,ring           ; routing decision.
        LDI GP5,[GP0]           ; next 2 insts. moved.-- destination
        ANI ICR,ICR,#----       ; release the bus.
        STLE GP7,GP4,BDRN       ; LE => GP7 =1 else GP7 =0.
        BNEZ GP7,ring           ; Broadcast transmission.
        MVI GP1,#----           ; address for decision table.
        MVI GP6,#----           ; command word.
        ST [GP1+#0],GP6         ; get the information regarding the
        ST [GP1+#1],GP5         ; destination address.
        LD GP6,[GP1+#2]         ; load the status.
        BSET GP6,#-,pending     ; the destination is busy.
        NOP                     ;
        NOP                     ;
indone: ANI GP7,GP6,#----       ; check for failed count.
        BEQZ GP7,error          ; The routing had been unsuccessful to this
        NOP                     ; destination.
        NOP                     ;
dllink: ORI SR2,SR2,#----       ; to indicate the Flood routing.
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        MVI GP7,#----           ; moved .. status to indicate flood routing.
        MVI GP0,#----           ; overwrite status info. from HSI.
        ST [GP0+#0],GP7         ; indicate the type to TR.
        ANI ICR,ICR,#----       ; release the bus.
        ORI ICR,ICR,#----       ; send interrupt to Token Ring.
        MVI GP7,#----           ; status to be written in the decision table.
        MV DEST,GP5             ;
        ORI SR1,SR1,#----       ; to indicate "I am busy"
        ADI SR2,SR2,#1          ; increment the no. of ongoing communications.
        JMP switch              ;
        ANI SR1,SR1,#----       ; moved .. reset Tx as being serviced.
ring:   ORI SR2,SR2,#----       ; to indicate the Broadcast Transmission.
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        ADI SR2,SR2,#1          ; increment the number of ongoing communications
        MVI GP0,#----           ; overwrite status info. from HSI.
        MVI GP7,#----           ; status to indicate broadcast transmission.
        ST [GP0+#0],GP7         ; indicate the type to TR.
```

```
        ANI ICR,ICR,#----       ; release the bus.
        ANI SR1,SR1,#----       ; reset Tx being serviced.
        ORI ICR,ICR,#----       ; send interrupt to Token Ring.
        MV DEST,GP5             ;
        JMP switch              ;
        ORI SR1,SR1,#----       ; moved ...indicate "I am busy".
error:  MVI GP1,#----           ; error code.
        MVI GP0,#----           ; address in shared memory.
        ST [GP0+#0],GP1         ;
        MVI GP1,#----           ; address for DMAC
        MVI GP2,#----           ; length
        MVI GP3,#----           ; command word
        ST [GP1+#0],GP0         ;
        ST [GP1+#1],GP2         ;
        ST [GP1+#2],GP3         ;
        JMP switch              ;
        ORI ICR,ICR,#----       ; moved..send DREQ and general purpose
                                  interrupt to host.

/* routine to switch the register bank at the end of interrupt routine.
   This should be done in hardware. */
switch: SHR GP1,SR1,#-          ; move pending ints. to LSBs.
        BEQZ GP1,watloop        ;
        NOP                     ;
        NOP                     ;
        BRSET ICR,#-,P0         ; if my priority=0 go to P0.
        SHR GP0,ICR,#-          ; move priority of ints. to LSBs.
        AND GP0,GP1,GP0         ; if the priority of pending routine is zero
        BNEZ GP0,return         ; goto P0.
        NOP                     ;
        NOP                     ;
P0:     ORI SR1,SR1,#----       ; set register window
        JMP return              ;
watloop:ORI SR1,SR1,#----       ; No pending interrupts... return to wait loop.
return: RET                     ;

/* On reception of Tx Interrupt from the host. */
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,SR1,#----       ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,SR1,#----       ; set the other window.
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        NOP                     ; get the message header from the host memory.
        NOP                     ;
ahead:  MVI GP1,#----           ; address for DMAC.
        MVI GP0,#----           ; set address.
        MVI GP2,#----           ; set length.
        MVI GP3,#----           ; command word.
        ST [GP1+#0],GP0         ;
        ST [GP1+#1],GP2         ;
        ST [GP1+#2],GP3         ;
        ANI ICR,ICR,#----       ; release the bus.
        ORI ICR,ICR,#----       ; moved ..send DREQ.
        JMP switch              ; set the register window.
        NOP                     ;

/* On reception of interrupt from either Token Ring or Data Link */
/* This indicates either the update or error condition. */

        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,SR1,#----       ; set the resgister window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,SR1,#----       ; set the other window.
        ORI ICR,ICR,#----       ; send BREQ
updtct: BSET SR1,#-,updtct      ; wait until the bus is free.
        NOP                     ;
        NOP                     ;
ahead:  MVI GP0,#----           ; shared memory address for update.
        LDI GP2,[GP0]           ; the type of update.
        LDI GP3,[GP0]           ; the source
        LDI GP4,[GP0]           ; and the destination.
        LDI GP5,[GP0]           ; Type of Transmission
        ANI ICR,ICR,#----       ; release the bus.
        BSET GP2,#-,control     ; Is it a control message (CACK)
        BSET GP2,#-,dack        ; or DACK
        BSET GP2,#-,LMinfo      ; or Learning mode routing info.
        NOP                     ;
        NOP                     ;
        STEQ GP6,GP3,SADD       ; error routine - check if this is source.
        BNEZ GP6,updts          ;
        NOP                     ;
        STEQ GP6,GP4,SADD       ; or destination.
        BNEZ GP6,updts          ; Fall through for an intermediate node.
        MV GP3,GP4              ; move destination so as to use same routine.
                                  (updts). MOVED ahead.
        NOP                     ;
        JMP toloop              ;
        SBI SR2,SR2,#1          ; reduce the number of ongoing communications.
                                  (MOVED).
updts:  MVI GP1,#----           ; CAM address for decision table.
        MVI GP6,#----           ; command word.
        ST [GP1+#0],GP6         ; indicate that the routing to the destination
        ST [GP1+#1],GP4         ; node has failed.
        LD GP7,[GP1+#2]         ; reduce the FM count in the decision table.
        SHR GP6,GP7,#-          ; failed count in LSB.
        SBI GP6,GP6,#1          ;
        SHL GP6,GP6,#-          ;
        ORI GP7,GP7,GP6         ;
        ANI GP7,GP7,#----       ; remove the busy flag.
        MVI GP6,#----           ; command word.
        ST [GP1+#0],GP6         ;
        ST [GP1+#1],GP3         ;
        ST [GP1+#2],GP7         ;
        SBI SR2,SR2,#1          ; reduce the number of ongoing communications.
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        MVI GP1,#----           ; error code.
        MVI GP0,#----           ; address in shared memory.
        ST [GP0+#0],GP1         ;
        MVI GP1,#----           ; address for DMAC
        MVI GP2,#----           ; length
        MVI GP3,#----           ; command word.
        ST [GP1+#0],GP0         ;
        ST [GP1+#1],GP2         ;
        ST [GP1+#2],GP3         ;
        JMP toloop              ;
        ORI ICR,ICR,#----       ; moved.. send DREQ and general purpose int.
control:STEQ GO6,GP3,SADD       ; am I source ?
        BNEZ GP6,set            ;
        NOP                     ;
        STEQ GP6,GP4,SADD       ; am I destination?
        BEQZ GP6,cont1          ; If not destination, go to cont1.
        NOP                     ;
```

```
set:    ORI SR1,SR1,#----       ; set "i am busy"
        JMP cont2               ;
cont1:  MVI GP1,#----           ; CAM address for decision table.
        MVI GP6,#----           ; command word.
        ANI GP5,GP5,#----       ; remove the busy flag.
        ST [GP1+#0],GP6         ; write info. for both s & d at intermediate
        ST [GP1+#1],GP4         ; node.
        ST [GP1+#2],GP5         ;
cont2:  MVI GP6,#-----          ; command word.
        ANI GP5,GP5,#----       ; remove the busy flag.
        ST [GP1+#0],GP6         ; and only for source at destination.
        ST [GP1+#1],GP3         ;
        ST [GP1+#2],GP5         ;
        ORI ICR,ICR,#----       ; send BREQ
toloop: BSET SR1,#-,toloop      ; look for more updates.
        NOP                     ;
        NOP                     ;
        LDI GP6,[GP0]           ;
        MVI GP7,#fffffh         ; if the next address has 0xffff there are
        STEQ GP7,GP6,GP7        ; no more entries.
        BNEZ GP7,updtct         ;
        NOP                     ;
        NOP                     ;
        JMP switch              ; switch the register bank.
        NOP                     ;
dack:   STEQ GP6,GP4,SADD       ; check for destination.
        BNEZ GP6,dackd          ;
        STEQ GP6,GP3,SADD       ; check for source.
        NOP                     ;
        BNEZ GP6,dacks          ;
        MVI GP1,#----           ; CAM address for decision table.
        MVI GP6,#---            ; command word.
        ANI GP5,GP5,#----       ; remove the busy flag.
        ST [GP1+#0],GP6         ; write info. for both s & d at intermediate
        ST [GP1+#1],GP4         ; destination.
        ST [GP1+#2],GP5         ;
        ST [GP1+#0],GP6         ; and source.
        ST [GP1+#1],GP3         ;
        JMP toloop              ;
        ST [GP1+#2],GP5         ; moved...
dacks:  MVI GP1,#----           ; CAM address for decision table.
        MVI GP6,#---            ; command word.
        ANI GP5,GP5,#----       ; data to be written .. remove the busy flag
        ST [GP1+#0],GP6         ; for destination.
        ST [GP1+#1],GP4         ;
        JMP toloop              ;
        ST [GP1+#2],GP5         ; moved...
dackd:  MVI GP1,#----           ; CAM address for decision table.
        MVI GP6,#----           ; command word.
        ANI GP5,GP5,#----       ; data to be written .. remove the busy flag
        ST [GP1+#0],GP6         ; for source.
        ST [GP1+#1],GP3         ;
        JMP toloop              ;
        ST [GP1+#2],GP5         ; moved...
LMinfo: STEQ GP6,GP3,SADD       ; source.
        BNEZ srccal             ;
        NOP                     ;
        STEQ GP6,GP4,SADD       ; destination.
        BNEZ destcal            ;
        NOP                     ;
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SP1,#-,wait        ; wait for the bus.
        MVI GP1,#----           ; address where status info. for DL is stored.
        MVI GP2,#----           ; status info.
        ST [GP1+#0],GP2         ;
        MV GP2,TABLE            ; The next available address where routing
        MV GP6,GP0              ; info. can be stored.
        MV GP7,GP5              ;
morechk:LDI GP1,[GP0+#0]        ; Intermediate address will be somewhere in
        STEQ GP1,GP1,SADD       ; between the list. Routing info. to two nodes
        BNEZ GP1,add1           ; (source and dest.) is therefore available for
        NOP                     ; an intermediate node.
        NOP                     ;
        SBI GP5,GP5,#1          ; go throught the whole information to find
        BNEZ GP5,morechk        ; the routing information.
        ADI GP0,GP0,#1          ; MOVED..
        NOP                     ;
        JMP dack                ;
        NOP                     ;
add1:   MV GP6,GP0              ;
cont:   LDI GP1,[GP0+#0]        ; First match has occured. Copy upto the first
        ST [GP2+#0],GP1         ; match. Then store the address in memory for
        SBI GP7,GP7,#1          ; DL. and then continue with second add.
        STEQ GP1,GP5,GP7        ;
        BNEZ GP1,cont           ;
        SBI GP0,GP0,#1          ; MOVED.
        ADI GP2,GP2,#1          ; MOVED.
        MV GP0,GP6              ;
        SBI GP0,GP0,#2          ;
        LD [GP0+#0],GP6         ; one address to the node.
        MVI GP1,#----           ; add. in shared memory for DL.
        ST GP6,[GP1+#0]         ;
        ADD GP0,GP0,#2          ;
        LD [GP0+#0],GP6         ; other address to the node.
        MVI GP1,#----           ; add. in shared memory for DL.
        ST GP6,[GP1+#0]         ;
        MV GP6,GP2              ;
cont1:  LDI GP1,[GP0+#0]        ;
        ST [GP2+#0],GP1         ;
        SBI GP7,GP7,#1          ;
        BEQZ GP7,cont1          ;
        ADI GP0,GP0,#1          ; MOVED.
        ADI GP2,GP2,#1          ; MOVED.
        ANI ICR,ICR,#----       ; release the bus.
        ORI ICR,ICR,#----       ; send interrupt to DL.
        MVI GP1,#----           ; CAM address for routing table.
        MVI GP5,#----           ; command word
        MVI GP0,#----           ; address for decision table.
        MVI GP7,#----           ; command word.
        ST [GP1+#0],GP5         ;
        ST [GP1+#1],GP3         ;
        ST [GP1+#2],GP2         ;
        MVI GP2,#----           ; indicate that the routing info. is learned.
        ST [GP0+#0],GP7         ; to the decision table for source.
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP2         ;
        ST [GP1+#0],GP5         ; to the routing table for dest.
        ST [GP1+#1],GP4         ;
        ST [GP1+#2],GP6         ;
        ST [GP0+#0],GP7         ; to the decision table for dest.
        ST [GP0+#1],GP4         ;
        JMP dack                ;
        ST [GP0+#2],GP2         ; moved.
srccal: ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SP1,#-,wait        ; wait for the bus.
        MVI GP1,#----           ; address where status info. for DL is stored.
        MVI GP2,#----           ; status info.
        ST [GP1+#0],GP2         ;
```

```
        MV GP2,TABLE            ; add. where routing table to be stored.
        ADD GP0,GP0,GP5         ; Add length and go to the end of table.
        LD GP3,[GP0+#0]         ; the address of the node next to source.
        ST [GP1+#1],GP3         ; store it for DL.
        MV GP7,GP5              ;
savemre:LD GP3,[GP0+#0]         ; load the address of the node.
        ST [GP2+#0],GP3         ; store it at new place.
        SBI GP5,GP5,#1          ; reduce the length
        ADI GP2,GP2,#1          ;
        SBI GP0,GP0,#1          ;
        BNEZ GP5,savemre        ;
        MVI GP1,#----           ; CAM address for routing table.
        NOP                     ;
        ANI ICR,ICR,#----       ; release the bus.
        ORI ICR,ICR,#----       ; send interrupt to DL.
        MVI GP5,#----           ; command word.
        ST [GP1+#0],GP5         ;
        ST [GP1+#1],GP4         ;
        JMP dackd               ;
        ST [GP1+#2],TABLE       ; moved...
        MV TABLE,GP7            ; new address.
destcal:ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ; wait for the bus
        MVI GP1,#----           ; address where status info. for DL is stored.
        MVI GP2,#----           ; status info.
        ST [GP1+#0],GP2         ;
        MV GP2,TABLE            ; add. where routing table to be stored.
        LD GP3,[GP0+#1]         ; the address of the node next to source.
        ST [GP1+#1],GP3         ; store it for DL.
        MV GP7,GP5              ;
savemre:LD GP4,[GP0+#0]         ; load the address of the node.
        ST [GP2+#0],GP4         ; store it at new place.
        SBI GP5,GP5,#1          ; reduce the length
        ADI GP2,GP2,#1          ;
        ADI GP0,GP0,#1          ;
        BNEZ GP5,savemre        ;
        MVI GP1,#----           ; CAM address for routing table.
        NOP                     ;
        ANI ICR,ICR,#-----      ; release the bus.
        ORI ICR,ICR,#----       ; send interrupt to DL.
        MVI GP5,#----           ; command word.
        ST [GP1+#0],GP5         ;
        ST [GP1+#1],GP3         ;
        ST [GP1+#2],TABLE       ; moved...
        JMP dacks               ;
        ADD TABLE,TABLE,GP7     ; new address.

/* this routine is a waiting loop. While waiting it does the book-keeping
   and initializes Fixed Routing or Learning Mode routing. */
pgwait: BSET SR2,#-,pgwait      ; check if initialization is done.
        NOP                     ;
        NOP                     ;
        ORI SR1,SR1,#----       ; set the register window.
again:  BSET SR1,#-,startLM     ; start Learning mode if its pending.
        BSET SR1,#-,startFR     ; start Fault routing if its pending.
        MV GP0,LDC              ; get the last address checked.
        NOP                     ;
        ADI LDC,LDC,#1          ; increment the address.
        LD GP1,[GP0+#0]         ; get the destination address.
        MVI GP2,#----           ; address of CAM.
        MVI GP3,#----           ; command word;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],GP1         ;
        LD GP4,[GP2+#2]         ;
        BSET GP4,#-,initdn      ;
        ANI GP5,GP4,#----       ; check if the learning count = 0;
        NOP                     ;
        BNEZ GP5,lm             ;
        NOP                     ;
        NOP                     ;
        JMP again               ;
initdn: ANI GP5,GP4,#----       ; check if the failed count = 0;
        BNEZ GP5,fr             ;
        NOP                     ;
        NOP                     ;
        JMP again               ;
lm:     MV LMPD,GP1             ;
        BSET GP4,#-,putpend     ; destination busy.
        MVI GP3,#----           ;
        BSET SR1,#-,putpend     ; source busy.
        ANI GP6,SR2,#----       ; determine the ongoing comms.
        BNEZ GP6,putpend        ; If there is no load on the network...
        NOP                     ;
        NOP                     ;
startLM:ORI SR1,SR1,#----       ; indicate learning mode routing as serviced.
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP2,#----           ; shared memory address.
        MVI GP3,#----           ; status
        MVI GP4,#----           ; length
        ST [GP2+#0],LMPD        ; Destination.
        ST [GP2+#1],GP3         ;
        ST [GP2+#2],GP4         ;
        ANI SR1,SR1,#----       ;
        ORI ICR,ICR,#----       ; send interrupt to TR
        MVI GP2,#----           ; address of CAM for decision table.
        MVI GP3,#----           ; command word;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],LMPD        ;
        LD GP4,[GP2+#2]         ;
        SHR GP5,GP4,#-          ; bring the LM count to LSBs
        SBI GP5,GP5,#1          ;
        SHL GP5,GP5,#-          ;
        ANI GP4,GP4,#----       ; make the LM field = 0
        ORI GP4,GP4,GP5         ; and replace with a new count
        ANI GP4,GP4,# ---       ; set it to busy.
        MVI GP3,#----           ; command word;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],LMPD        ;
        ST [GP2+#2],GP4         ;
        JMP again               ;
        ANI ICR,ICR,#----       ; moved... reset the LM service flag.
putpend:ORI SR1,SR1,# ---       ; put learning mode in pending list.
        JMP again               ;
        NOP                     ;
fr:     MV FRPD,GP9             ;
        BSET SR1,#-,putpnd2     ; source busy.
        MVI GP3,#----           ;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],GP1         ;
        ST [GP2+#2],GP4         ;
        BSET GP4,#-,putpnd2     ; destination busy.
        ANI GP6,SR2,#- ---      ; determine the ongoing comms.
        BNEZ GP6,putpnd2        ;
        NOP                     ;
        NOP                     ;
startFR:ORI SR1,SR1,#----       ; indicate flood routing is being serviced.
        MVI GP2,#----           ; address in CAM for routing table.
```

```
        MVI GP3,#----           ; command
        ORI ICR,ICR,#----       ; send BREQ
wait:   BSET SR1,#-.wait        ;
        MVI GP2,#----           ; shared memory address.
        MVI GP3,#----           ; status
        MVI GP4,#----           ; length
        ST [GP2+#0],FRPD        ;
        ST [GP2+#1],GP3         ;
        ST [GP2+#2],GP4         ;
        ST [GP2+#3],GP1         ; routing info. address in shared memory.
        ANI SR1,SR1,#----       ;
        ORI ICR,ICR,#----       ; send interrupt to TR
        MVI GP2,#----           ; address of CAM for decision table.
        MVI GP3,#----           ; command word;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],FRPD        ;
        LD GP4,[GP2+#2]         ;
        SHR GP5,GP4,#-          ; bring the FR count to LSBs
        SBI GP5,#1              ;
        SHL GP5,GP5,#-          ;
        ANI GP4,GP4,#----       ; make the FR field =  0
        OR GP4,GP4,GP5          ; and replace it with a new count.
        ANI GP4,GP4,#----       ; set it to busy.
        MVI GP3,#----           ; command word;
        ST [GP2+#0],GP3         ;
        ST [GP2+#1],FRPD        ;
        ST [GP2+#2],GP4         ;
        JMP again               ;
        ANI ICR,ICR,#----       ; moved... reset the FR service flag.
putpnd2:ORI SR1,SR1,#----       ; put fixed mode mode in pending list.
        JMP again               ;
        NOP                     ;

/* end od protocol implementation for data processor */
```

```
/* ------------------------------------------------------------------------
                              TOKEN RING INTERFACE
   ------------------------------------------------------------------------ */

/* This is the sample implementation of the protocol functions carried out in
   the token ring interface. The code is not complete or error free. It is not
   executed on any particular processor.
   However it has helped in determining the register and instruction
   requirements. This exercise will also allow us to estimate the performance.
   The control hazards that occur in pipeline architecture are taken care of.
   (MOVED in comments indicates that an instruction is rescheduled to avoid
   control hazard.
*/

/* The code indicates the register required for each function. The exact bit
   position is postponed until final implementation. */

/* At PG2 interrupt--- Initialization routine. */

        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; check if the local bus is held by other proc.
        MVI GP1,#----           ; address for DMAC
        MVI GP0,#----           ; set address.
        MVI GP2,#----           ; set length (known).
        MVI GP3,#----           ; command word
        ST [GP1+#0],GP0         ; store the address and length in
        ST [GP1+#1],GP2         ; DMAC (host system interface) registers.
        ST [GP1+#2],GP3         ;
        ORI CR,#----            ; send DREQ.
waitTC: BSET SR1,#-,waitTC      ; wait for TC.
        NOP                     ;
        NOP                     ;
        LDI INTDP,[GP0]         ; Interrupt vectors...
        LDI INTDL,[GP0]         ; From Data Link Interface.
        LDI INTTHT,[GP0]        ; From token holding timer.
        LDI INTTNT,[GP0]        ; From no token timer.
        LDI INTMSG,[GP0]        ; Interrupt due to either C/D message.
        LDI THT,[GP0]           ; token holding value.
        LDI TNT,[GP0]           ; no token, hence generate value.
        LDI SADD,[GP0]          ; source address.
        LDI SDEL,[GP0]          ; starting delimier...
        LDI EDEL,[GP0]          ; ending delimiter..
        LDI LADD,[GP0]          ; last address used to store routing info.
        ANI CR,#----            ; release the bus.
        ORI SR1,#----           ; initialization done.
        RET
        ORI ICR,#----           ; Interrupt enable. (MOVED)

/* On interrupt from Data Processor */
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        NOP                     ;
        NOP                     ;
        JMP ahead               ;
        ORI SR1,#----           ; set the register window. (MOVED)
zero:   ORI SR1,#----           ; set the other window.
ahead:  ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ; wait for the free bus.
        MVI GP0,#----           ; address of the shared memory.
        NOP                     ;
        LDI GP1,[GP0]           ; load the status.
        LDI DADD,[GP0]          ; destination.
        LDI LENGTH,[GP0]        ; length of the message.
        LDI PRIO,[GP0]          ; Priority.
        ANI CR,#----            ; release the bus.

/* The interrupt from the data processor can be due to... */
        BSET GP1,#-,frTX        ; flood routing transmission.
        BSET GP1,#-,lmTX        ; learning mode routing.
        BSET GP1,#-,ftTx        ; fault routing.
        NOP                     ;
        NOP                     ;

/* Broadcast transmission.... */
        ORI CR,#----            ; send BREQ.
brdTx:  BSET SR1,#-,brdTx       ;
        MVI GP1,#----           ; add. of DMAC.
        MVI GP0,#----           ; add. of FIFO.
        MVI GP3,#----           ; command word.
        ST [GP1+#0],GP0         ; Load the address and length in DMAC reg.
        ST [GP1+#1],LENGTH      ;
        ST [GP1+#2],GP3         ;
        ORI CR,#----            ; send DREQ.
        MVI AC,#----            ; load access control info.
        MVI FC,#----            ; frame control. (MOVED)
        MVI FS,#----            ; frame status. (MOVED)
        ORI CR,#----            ; start Tx.
        ORI SRC,#----           ; indicate source as busy.
        ORI DEST,#-----         ; don't forward the message.
waitTC: BRSET SR1,#-,waitTC     ; wait for TC.
        NOP                     ;
        NOP                     ;
        ANI CR,#----            ; release the bus.
waitMsg:BRSET SRC,#-,waitMsg    ; wait for the whole message to be transmitted.
        NOP                     ;
        NOP                     ;
waitSD: BRSET DEST,#-,waitSD    ; wait for Starting delimiter to return.
        MVI AC,#----            ; free token, AC frame.
        NOP                     ;
        ORI SRC,#----           ; release token.
waitED: BRSET DEST,#-,waitED    ; wait for Ending delimiter to return.
        MVI GP0,#----           ;
        NOP                     ;
        ANI DEST,#----          ; enable forwarding of message.
        AND GP0,FS,GP0          ; check for error in the received frame.
        BNEZ GP0,noerr          ;
        NOP                     ;
        MVI GP1,#----           ; indicate an error.
        JMP wait                ;
        NOP                     ;
        NOP                     ;
noerr:  MVI GP1,#----           ; status indicating no error.
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        NOP                     ;
        MVI GP0,#----           ; pass on the status to DP.No need to transmit.
        ST [GP0+#0],DADD        ; destination.
        ANI CR,#----            ; release the bus.
        JMP setwind             ; set the register window.
        ORI ICR,#----           ; send interrupt to DP. (MOVED)

/* flood routing initiation at source... */
frTX:   MVI AC,#----            ; load access control info.
        ORI SR2,#----           ; type of Tx.
cont:   ORI SRC,#----           ; indicate the source as busy.
        MVI FC,#----            ; frame control. (MOVED)
        MVI FS,#----            ; frame status.  (MOVED)
        ORI CR,#----            ; start Tx.
        ORI DEST,#----          ; Don't forward the message.
        MVI GP0,#----           ; Address in Shared Memory.
```

```
waitED: BRSET DEST,#-,waitED    ; wait for Ending delimiter to return.
        MVI GP1,#----           ; set mask. (MOVED)
        AND GP1,SR1,GP1         ; set the type of Tx. (MOVED)
        BSET FS,#-,destbsy      ;
        NOP                     ;
        NOP                     ;
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        NOP                     ;
        NOP                     ;
        ST [GP0+#0],GP1         ; send the type of Tx.
        ST [GP0+#1],DADD        ;
        ST [GP0+#2],LENGTH      ;
ahead:  ORI ICR,#----           ; send interrupt to Data Link.
        JMP setwind             ; set the register bank.
        ANI DEST,#----          ; disable forwarding of message. (MOVED)

/* if the destination is busy..*/
destBSY:BSET SR1,#-,lmBSY       ; The destination node is busy. Postpone the
        BSET SR1,#-,ftBSY       ; transmission.
        MVI AC,#----            ; free token AC frame.
        NOP                     ;
        ORI SRC,#----           ; release token.
        MV BUSYDST,DADD         ;
        JMP setwind             ;
        ORI SR2,#----           ; indicate in SR2 that destination is busy.(M)
lmBSY:  MVI GP1,#----           ; indicate DP about busy destination.
        JMP ahead               ;
        NOP                     ;
ftBSY:  MVI GP1,#----           ;
ahead:  ORI CR,#----            ;
        BSET SR1,#-,wait        ;
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0+#0],GP1         ;
        JMP setwind             ;
        ORI ICR,#----           ; send interrupt to DP. (MOVED)

/* Learning mode routing initiation at source .. */
lmTX:   MVI AC,#----            ; load access control info.
        JMP cont                ; the routine is same as flodd routing.
        ORI SR2,#----           ; type of Tx. (MOVED)

/* Fault routing initiation at source .. */
ftTX:   MVI AC,#----            ; load access control info.
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ; flood routing reads the routing info.
        MV GP0,LENGTH           ; from the shared memory and writes it
        NOP                     ; into the transmit FIFO.
        LDI GP3,[GP0]           ;
        WFIFO GP3               ;
moredta:BNEZ GP2,frd            ;
        NOP                     ;
        NOP                     ;
        WFIFO GP1               ;
        LDI GP1,[GP0]           ;
        STEQ GP2,DADD,GP1       ;
        BEQ2 GP2, moredta       ;
        NOP                     ;
        NOP                     ;
frd:    JMP cont                ; The remaining routine remains the same as
        NOP                     ; flood routing.

/* on interrupt due to message received (RFIFO not empty). */

        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        NOP                     ;
        ORI SR1,#----           ; set the register window.
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
intmsg: BRSET DEST,#-,cntlRx    ; If control message go to cntlRx.
watEDEL:BRSET DEST,#-,watEDEL   ; else its a data message due to broadcast Tx.
        NOP                     ; wait for EDEL of the data message.
        NOP                     ;
        BRSET DEST,#-,noerr     ; is there any error ?
        NOP                     ;
        NOP                     ;
        JMP setwind             ; return.
        ORI CR,#----            ; reset FIFO. (MOVED)
/* Its the data message due to broadcast transmission. Already checked for
   error. */
noerr:  ORI ICR,#----           ; enable interrupts.
        RFIFO GP4               ; The MAC control frames are read into registers
        RFIFO GP5               ;
        RFIFO DADD              ;
        SBI RLENGTH,#5          ; The data message length.
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ; wait for the bus.
        MVI GP6,#----           ; address of DMAC.
        MVI GP4,#----           ; address of FIFO.
waitTC: BRSET SR1,#-,waitTC     ; wait for TC.
        ST [GP5+#0],GP6         ; (MOVED)
        ST [GP5+#1],RLENGTH     ; the size of data in FIFO. (MOVED)
        ANI CR,#----            ; release the bus.
        RFIFO GP4               ;
        JMP setwind             ; set the register bank.
        RFIFO GP4               ; CRC. (MOVED)

/* The message received is a control message */
cntlRx: RFIFO GP2               ; control info.
        RFIFO GP3               ; destination.
        RFIFO GP4               ; source.
        BSET GP2,#-,prfr        ; flood routing.- path request.
        BSET GP2,#-,prlm        ; learning mode routing. - path request.
        BSET GP2,#-,prft        ; fault routing. - path request.
        BSET GP2,#-,pgfr        ; flood routing - path grant.
        BSET GP2,#-,pglm        ; learning mode - path grant.
        BSET GP2,#-,pgft        ; fault routing - path grant.
        BSET GP2,# ,prls        ; flood routing - path release.
        NOP                     ;
        NOP                     ;
/* This is the path request message for the flood routing. Its received at
   destination and intermediate nodes. */
prfr:   STEQ GP7,GP3,SADD       ; check for destination.
        BEQ2 GP7,cont           ;
        NOP                     ;
        NOP                     ;
        MV BUSYDST,DADD         ;
        ORI SR2,#----           ; indicate in SR2 that destination is busy.
cont:   ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ; wait for the bus.
        MV CSRC,GP4             ;
        MV CDEST,GP3            ;
        MVI GP0#----            ; load the status for the use by data link
        ST [GP0#0],GP2          ; and data processor.
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP4         ;
        ANI CR,# ---            ; release the bus.
```

```
        ORI ICR,#----           ; send interrupt to DL.
        STEQ GP7,GP3,SADD       ;
        BEQZ GP7,cont2          ;
        NOP                     ;
        NOP                     ;
        ORI ICR,#----           ; send interrupt to DP.
cont2:  JMP setwind             ;
        NOP                     ;
/* This is the path request message for the fault routing. This includes the
   list of the nodes which will be in the path. */
prft:   STEQ GP0,GP3,SADD       ; check if the control msg. is for destination.
        BEQZ GP0,contft         ;
        MVI GP1,#-----          ;
        NOP                     ;
        MV DADD,GP4             ;
        ORI SRC,#----           ;
        MV BUSYDST,DADD         ;
        ORI SR2,#----           ; indicate in SR2 that destination is busy.
        RFIFO GP5               ;
        JMP ahead               ;
        MV GP6,GP5              ;
contft: MVI GP1,#----           ; destination for fault routing.(MOVED)
        MV CSRC,GP4             ;
        MV CDEST,GP3            ;
contnxt:RFIFO GP5               ;
        STEQ GP7,GP5,SADD       ; if no, get the next word.
        BEQZ GP7,next           ; check if this belongs to intermediate node.
        NOP                     ;
        NOP                     ;
        BRSET SR2,#-,cont       ; if yes, get the immediate addresses for
        NOP                     ;
        MV GP6,GP4              ; data and control messages. All the time
        JMP ahead               ; checking that the FIFO is not empty. Two
        NOP                     ;
cont:   RFIFO GP6               ; words are stored at any time as the words
        JMP ahead               ; (addresses)on either side of the node are rqd.
        NOP                     ;
next:   BSET SR2,#-,nopath      ; FIFO empty.
        NOP                     ;
        NOP                     ;
        RFIFO GP6               ;
        STEQ GP7,GP6,SADD       ;
        BEQZ GP7,check          ;
        MV GP5,GP6              ;
        NOP                     ;
        BRSET SR2,#-,cont2      ;
        MV GP6,GP4              ;
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
cont2:  RFIFO GP6               ;
        JMP ahead               ;
        NOP                     ;
check:  BSET SR2,#-,nopath      ; check if FIFO is empty.
        NOP                     ;
        NOP                     ;
        JMP contnxt             ;
        NOP                     ;
        ORI CR,#----            ;
ahead:  BSET SR1,#-,ahead       ; send BREQ.
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0+#0],GP1         ; If the node is in the path, indicate to the
        ST [GP0+#1],SADD        ; data link interface.
        ST [GP0+#2],DADD        ;
        ST [GP0+#3],GP5         ;
        ST [GP0+#4],GP6         ;
        ANI CR,#----            ;
        ORI ICR,#----           ; release the bus.
nopath: ORI CR,#----            ; interrupt to data link interface.
wait:   BSET SR1,#-,wait        ; send BREQ.
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0+#0],GP1         ;
        ST [GP0+#1],GP2         ;
        ST [GP0+#2],GP3         ;
        ANI CR,#----            ;
        JMP setwind             ; release the bus.
        ORI ICR,#----           ;
                                ; interrupt to DP interface. (MOVED)
/* This is the path grant or path-setup for flood routing which is received at
   source node and intermediate nodes. */
pgfr:   STEQ GP7,GP3,SADD       ;
        BNEZ GP7,rlstkn         ;
        NOP                     ;
        NOP                     ;
/* at intermediate node... */
        ORI CR,#----            ;
wait:   BSET SR1,#-,wait        ; send BREQ.
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0#0],GP2          ;
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP4         ;
        ANI CR,#----            ;
        JMP chkbsy              ; release the bus.
        ORI ICR,#----           ;
/* at source node... */         ; send interrupt to DP. (MOVED)
rlstkn: MVI AC,#----            ;
        NOP                     ; free token, AC frame.
        ORI SRC,#----           ;
        BSET GP2,#-,prls        ; release token.
        NOP                     ; if its fault routing service.....
        NOP                     ;
        ORI CR,#----            ;
wait:   BSET SR1,#-,wait        ; send BREQ.
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0#0],GP2          ;
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP4         ;
        ANI CR,#----            ;
        ORI ICR,#----           ; release the bus.
        JMP setwind             ; send interrupt to DP. (MOVED)
        NOP                     ;
/* this is the message received at source or intermediate node. This indicates
   end of current transmission. */
prls:   ORI CR,#----            ;
wait:   BSET SR,#-,wait         ; send BREQ.
        MVI GP0,#----           ;
        NOP                     ;
        ST [GP0+#0],GP2         ;
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP4         ;
        ANI CR,#----            ;
        ORI ICR,#----           ; release the bus.
chkbsy: STEQ GP7,GP2,BUSYDST    ; send interrupt to DP.
```

```
        BEQZ GP7,search         ;
        NOP                     ;
        NOP                     ;
        JMP ahead               ;
        ANI SR1,#----           ; busy destination. (MOVED)
search: STEQ GP7,GP3,BUSYDST    ;
        BEQZ GP7,ahead          ;
        NOP                     ;
        NOP                     ;
ahead:  JMP setwind             ;
        ANI SR1,#---            ; reset the destination unavailable flag. (M)

/* routine for the path grant or path setup message. This indicates successful
   testing. Its routines are same as for flood routing.*/
pgft:   STEQ GP7,GP3,SADD       ; check for the source or intermediate node */
        BNEZ GP7,rlstkn         ;
        NOP                     ;
        NOP                     ;
/* Intermediate node */
backft: CALL prls               ; reset the allocated links.
        NOP                     ;
        JMP setwind             ;
        NOP                     ;

/* path request for learning mode .. */
prlm:   STEQ GP7,GP3,SADD       ; check for destination or intermediate node.
        BEQZ GP7,intlm          ;
        NOP                     ;
        NOP                     ;
/* destination node */
        MV BUSYDST,DADD         ;
        ORI SR1,#----           ; Indicate destination busy.
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        MV CSRC,GP4             ;
        MV CDEST,GP3            ;
intlm:  MVI GP0,#----           ; inform the data link interface and data
        ST [GP0#0],GP2          ; processor about the routing.
        ST [GP0+#1],GP3         ;
        ST [GP0+#2],GP4         ;
        ANI CR,#----            ; release the bus.
        ORI ICR,#----           ; send interrupt to DL.
        STEQ GP7,GP3,SADD       ;
        BEQZ GP7,cont2          ;
        NOP                     ;
        NOP                     ;
        ORI ICR,#----           ; send interrupt to DP.
cont2:  JMP setwind             ;
        NOP                     ;

/* path grant or path setup message for the learning mode.. It saves the
   routing info. into the memory and indicates this add. to the data processor.
*/
pglm:   ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP0,#----           ;
        MVI GP1,#----           ; status indicating LM.
        ST [GP0+#0],GP1         ;
        ST [GP0+#1],CSRC        ;
        ST [GP0+#2],CDEST       ;
        ST [GP0+#3],RLNTH       ; length of the routing info.
        ADI GP0,#0004h          ;
moredta:RFIFO GP2               ; store the routing info. into the shared mem.
        BNEZ RLNTH,moredta      ;
        ST [GP0+#0],GP2         ; MOVED.
        NOP                     ;
        JMP setwind             ;
        ANI CR,#----            ; MOVED.

/* Interrupt from Data Link .... CACK or DACK.
   At this point send the path setup or path release message.
*/
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,#----           ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
        ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        MVI GP0,#-----          ;
        NOP                     ;
        LDI GP1,[GP0]           ;
        LDI SADD,[GP0]          ;
        LDI DADD,[GP0]          ;
        ANI CR,#----            ; release the bus.
        BSET GP1,#-,dack        ;
        NOP                     ;
        NOP                     ;
/* path setup message */
        BSET GP1,#-,noLM        ;
        NOP                     ;
        NOP                     ;
        LDI GP2,[GP0]           ;
cont:   BEQZ GP2,ahead          ;
        NOP                     ;
        NOP                     ;
        LDI GP3,[GP0]           ;
        WFIFO GP3               ;
        JMP cont                ;
        SBI GP2,#1              ;
ahead:  ANI CR,#----            ; release the bus.
        ORI ICR,#----           ; send Interrupt to DP.
noLM:   SHR GP2,SR2,#-          ; Shifted .. so that can be used in AC frame.
        ANI GP2,#----           ;
        MV AC,GP2               ;
        MVI FC,#----            ;
        MVI FS,#----            ;
        ORI SRC,#----           ;
        ORI RCV,#----           ; start TX.
waitED: BRSET RCV,#-,waitED     ; don't forward the msg.
        NOP                     ;
        NOP                     ;
        ANI RCV,#----           ; enable forwarding of message.
        JMP setwind             ;
/* path release message */
dack:   MVI AC,#----            ;
        MVI FC,#----            ; MOVED.
        MVI FS,#----            ; MOVED.
        ORI SRC,#---            ; start TX.
        ORI RCV,#----           ; don't forward the msg.
waitED: BRSET RCV,#-,waitED     ;
        MVI AC,#----            ; free token frame.
        NOP                     ;
        ORI SRC,#----           ; release token.
        JMP setwind             ;
        ANI RCV,#---            ; enable forwarding of message.(MOVED)
```

```
/* Routine which sets up the proper register window at the time of returning.*/
setwind:SHR GP1,SR1,#-          ; move pending ints. to LSBs.
        BEQZ GP1,watloop        ;
        NOP                     ;
        NOP                     ;
        BRSET ICR,#-,P0         ; my priority
        NOP                     ;
        NOP                     ;
        SHR GP0,ICR,#-          ; move priority of ints. to LSBs.
        AND GP0,GP1,GP0         ;
        BNEZ GP0,return         ;
        NOP                     ;
        NOP                     ;
P0:     ORI SR1,#----           ; set register window.
        RET                     ;
        NOP                     ;
watloop:ORI SR1,#----           ;
return: RET                     ;
        NOP                     ;

/* end of protocol implementation for token ring interface */
```

```
/* ------------------------------------------------------------------
                        DATA LINK INTERFACE
   ------------------------------------------------------------------ */
/* This is the sample implementation of the protocol functions carried out in
   the data link interface. The code is not complete or error free. It is not
   executed on any particular processor.
   However it has helped in determining the register and instruction
   requirements. This exercise will alllow us to estimate the performance.
   The control hazards that occur in pipeline architecture are taken care of.
   (MOVED in comments indicates that an instruction is rescheduled to avoid
   control hazard.
*/

/* The code indicates the register required for each function. The exact bit
   position is postponed until final implementation. */

/* Initialization routine executed at PG3 interrupt. */
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        MVI GP1,#----           ; address for DMAC (Host system Interface).
        MVI GP0,#----           ; set address.
        MVI GP2,#----           ; set length (known).
        ST [GP1+#0],GP0         ; store the address and length in DMAC
        ST [GP1+#1],GP2         ; registers.
        ORI CR,#----            ; send DREQ.
pg3:    BRSET SR1,#-,pg3        ; wait until the TC is received.
        NOP                     ;
        NOP                     ;
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until bus is granted.
        NOP                     ;
        NOP                     ;
        LDI INTDP,[GP0]         ; Interrupt vectors...
        LDI INTTR,[GP0]         ; From token ring.
        LDI INTTM,[GP0]         ; From timer.
        LDI CACK,[GP0]          ; CACK received interrupt vector.
        LDI TIMER,[GP0]         ; timer register.
        LDI SADD,[GP0]          ; source address.
        LDI DLR,[GP0]           ; initialize max. hop count, no. of links and
                                  masks for links.
        LDI ICR,[GP0]           ; Initialize interrupt control registers.
        LDI MAXFM,[GP0]         ; define maximum frame size.
        LDI TABLE,[GP0]         ; address in memory mapping links to imm. destn.
        MV  GP0,TABLE           ;
        SHR GP1,DLR,#-          ; get number of links to LSBs.
        MVI GP2,#ffffh          ;
updt:   ST [GP0+#0],GP2         ; #ffffh implies, not initialized yet.
        SBI GP1,#1              ;
        BNEZ GP1,updt           ; Repeat this for all links.
        ADI GP0,#1              ; MOVED.
        NOP                     ;
        ANI CR,#----            ; release the bus.
        ORI SR1,#----           ; initialization done.
        RET                     ;
        ORI ICR,#----           ; Interrupt enable. (MOVED)

/* The interrupt from Token Ring */
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,#----           ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait till the shared bus is free.
        MVI GP0,#----           ; load the address to read from in the memory.
        NOP                     ;
        LDI GP1,[GP0]           ; get the status.
        ANI CR,#----            ; The interrupt is for..
        BSET GP1,#-,startFR     ; starting flood routing.
        BSET GP1,#-,contTx      ; sending data message via flood routing.
        BSET GP1,#-,startLM     ; start learning mode routing.
        BSET GP1,#-,LMsrc       ; receive the learned information.
        BSET GP1,#-,startFT     ; start Fault routing.
        BSET GP1,#-,FR          ; Flood routing at intermediate and destination.
        BSET GP1,#-,LM          ; LM routing at intermediate and destination.
        BSET GP1,#-,FT          ; Fault routing at intermediate and destination.
        BSET GP1,#-,cackTR      ; Token Ring received path grant message.
        NOP                     ;
        NOP                     ;
        JMP error               ; else there is an error.
        NOP                     ;
/* send a setup message on all available links */
startFR:ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        ORI SR1,#----           ; indicate the int. serviced.
        ORI SR2,#----           ; and type of Tx.
        LDI DADD,[GP0]          ; destination address.
        LDI LENGTH,[GP0]        ; length.
        ANI CR,#----            ; release the bus.
        SSTLN                   ; Set the available links for output.
        MV R0,AVLN              ;
        MV SELDST,AVLN          ; set selector to destination to receive CACK.
        MV RLENGTH,LENGTH       ;
        MVI CINFO,#----         ; control info.
        WFIFO GP1               ; Write a sample data in the FIFO.
        ORI SRC,#----           ; start Tx and indicate source as busy and set
                                  CCHK bit.
        ORI CR,#----            ; start timer
wait:   BRSET SRC,#-,wait       ; wait for the data Tx to complete.
        NOP                     ;
        NOP                     ;
waitack:BREST DEST,#-,waitack   ; wait for CACK from destination.
        NOP                     ;
        NOP                     ;
        SRLSLN                  ; Release unused links.
        MVI GP2,#ffffh          ;
        XOR GP2,R0,GP2          ; invert R0.
        MV SELDST,GP2           ; reset the destination selector on all
        JMP setwind             ; unused links.
        ANI CR,#- --            ; reset timer. (MOVED)
/* path grant message is received on token ring and CACK on data links.
   Send the data message now. */
contTx: ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP1,#----           ; DMAC address.
        MVI GP0,#----           ; address of FIFO.
        STLE GP6,RLENGTH,MAXFM  ; If this is the last frame..
        BNEZ GP6,cont           ; goto cont..
        NOP                     ;
        NOP                     ;
        MV GP2,MAXFM            ; else send messae of size = MAXFM.
        JMP ahead               ;
        SBB RLENGTH,RLENGTH,MAXFM ; MOVED.
cont:   MV GP2,RLENGTH          ;
        MVI RLENGTH,#0000h      ;
ahead:  ST [GP1+#0],GP0         ;
        ST [GP1+#1],GP2         ;
```

```
        ORI CR,#----            ; send DREQ.
loop:   BSET SRC,#-,loop        ; wait until FIFO is not empty.
        NOP                     ;
        NOP                     ;
        ORI SRC,#----           ; start Tx and indicate source as busy.
wait:   BRSET SRC,#-,wait       ; wait for the transmission to complete.
        NOP                     ;
        NOP                     ;
        BSET SR1,#-,moredta     ; If TC is received....
        NOP                     ;
        NOP                     ;
        JMP loop                ;
        NOP                     ;
moredta:BNEZ RLENGTH,contTx     ; If the RLENGTH <> 0 .. it indicates that
        NOP                     ; there is still message required to be
        NOP                     ; transmitted. Continue to send...
        ORI SRC,#----           ; set the DCHK bit.
wait:   BRSET DEST,#-,wait      ; check for DACK.
        MVI SELDST,#0           ; Reset the destination selector.
        NOP                     ;
        JMP setwind             ; switch the register bank.
        SRSTLN                  ; reset the used links. MOVED.
/* The messages received at intermediate and destination nodes. */
FR:     ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        ORI SR2,#----           ; set the type of Tx.
        ORI SR1,#----           ; and int. serviced.
        LD GP1,[GP0]            ; destination.
        LD GP2,[GP0]            ; source.
        ANI CR,#----            ; release the bus.
        STEQ GP3,SADD,GP1       ;
        BNEZ GP3,destFR         ; I am destination.. go to destFR.
        NOP                     ;
        NOP                     ;
        STLN                    ; switch available links for output.
        MV R0,AVLN              ;
        STRST @R0,#----         ; load the state =1 and value of the counter.
        MV INIT,R0              ; for all available links.
        ORI CR,#----            ; start timer
wait:   BSET DEST,#-,wait       ; wait for SDEL being detected on any one link.
        NOP                     ;
        NOP                     ;
        XOR R0,AVLN,USLN        ; When the message is received ...
        STRST @R0,# ---         ; set other links to state = 2.
        MV INIT,R0              ;
wait:   BSET DEST,#-,wait       ; wait for EDEL being detected.
        MV  R0,USLN             ;
        NOP                     ;
        JMP setwind             ; the messsage was received on R0..
        STRST,@R0,#----         ; set the link in state = 3. (MOVED)
destFR: MV DEST,GP1             ;
        SSTLN                   ; set the available links for input.
        MV R0,AVLN              ;
        MV SELDST,AVLN          ; set selector to destination.
        ORI CR,#----            ; start timer
waitdel:BSET DEST,#-,waitdel    ; for SDEL being detected.
        NOP                     ;
        NOP                     ;
        SRLSLN                  ;
        XOR R0,AVLN,USLN        ;
        XOR GP2,R0,#ffffh       ; invert R0.
        MV SELDST,GP2           ; reset the destination.
        ANI CR,#----            ; reset timer.
        ORI ICR,#----           ; enable EDEL interrupt.
```

```
        MVI CINFO,#----         ; CACK frame.
        ORI SRC,#----           ; start TX. send CACK on the link on ehich
                                ; message was received.
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait for the bus.
        MVI GP0,#----           ;
        MVI GP1,#----           ; write the control code.
        ST [GP0+#0],GP1         ;
        ORI ICR,#----           ; send interrupt to TR.
        JMP setwind             ;
        RFIFO GP2               ; sample data.. not important. (MOVED)

/* switch register banks. This routines should be carried out in hardware */
setwind:SHR GP1,SR1,#-          ; move pending ints. to LSBs.
        BEQZ GP1,watloop        ; If there are no pending interrupts.
        NOP                     ;
        NOP                     ;
        BRSET ICR,#-,P0         ; my priority=0 jump to P0.
        NOP                     ;
        NOP                     ;
        SHR GP0,ICR,#-          ; move priority of ints. to LSBs.
        AND GP0,GP1,GP0         ; If the pending interrupt has priority=0 goto
        BNEZ GP0,return         ; P0.
        NOP                     ;
        NOP                     ;
P0:     ORI SR1,#----           ; set register window
        RET                     ;
        NOP                     ;
watloop:ORI SR1,#----           ; switch the register bank for wait loop.
return: RET                     ;
        NOP                     ;

/* start learning mode routing at source */
startLM:ORI CR,# ---            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait until the bus is free.
        ORI SR2,#----           ; set the type of TX.
        ORI SR1,#----           ; and int. serviced.
        LDI DADD,[GP0]          ; destination address.
        ANI CR,#----            ; release the bus.
        SSTLN                   ; set up links for output.
        MV R0,AVLN              ;
        MV SELDST,AVLN          ; set selector to destination.
        MVI CINFO,#----         ;
        ORI SRC,#----           ; start Tx and indicate source as busy and set
                                  CCHK bit.
        ORI CR,#----            ; start timer
watcack:BRSET DEST,#-,watcack   ; wait until CACK is received.
        NOP                     ;
        NOP                     ;
        SRSTLN                  ; reset links.
        MVI SELDST,#0000h       ; reset selector to destination.
        ANI CR,#----            ; reset timer.
        MV ENCODE,USLN          ; save the used link for later updation.
        JMP setwind             ;
        ENC UPDT                ; encode 16->4  (MOVED). The link will
                                  have to be update when it receives info.
                                  from token ring.
/* This is from token ring to source.. indicating it to update the tables
  with the information learned during learning mode routing for the links. */
LMsrc:  ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ; wait until the bus is released.
        MVI GP0,#----           ;
        LDI GP1,[GP0]           ; the node address from which msg. was received.
        MVI GP2,#----           ;
```

```
        AND GP2,UPDT,GP2        ; mask the MSBs.
        MV DECODE,GP2           ;
        DEC GP4                 ; decode  4->16
        MV GP0,TABLE            ; add. in shared memory where the link address
                                  is stored.
        SHR GP3,DLR,#-          ; get the number of links.
        SBB GP4,GP3,GP4         ; its stored with highest link first...
        ADD GP0,GP0.GP4         ;
        JMP setwind             ;
        ST [GP0+#0],GP1         ; store the address at the link. (MOVED).
/* at the intermediate and destination nodes when the token ring receives path
   request message. */
LM:     ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        ORI SR2,#----           ; set the type of Tx.
        ORI SR1,#----           ; and int. serviced.
        LDI GP1,[GP0]           ; destination.
        LDI GP2,[GP0]           ; source.
        STEQ GP3,SADD,GP1       ; check for destination.
        BNEZ GP3,destLM         ;
        NOP                     ;
        NOP                     ;
        ANI CR,#----            ; release the bus.
        STLN                    ; set links.
        MV R0,AVLN              ; find available links and
        STRST @R0,#----         ; set them in state = 5.
wait:   BSET DEST,#-,wait       ; wait for SDEL being detected on any one link.
        MV INIT,R0              ; MOVED.
        NOP                     ; Once the message is received ..
        XOR R0,AVLN,USLN        ; determine unused links and
        STRST @R0,#----         ; set those links to state = 2.
        JMP setwind             ;
        MV INIT,R0              ; MOVED.
/* at the destination .. */
destLM: ANI CR,#----            ; release the bus.
        SSTLN                   ; set the links for input.
waitDEL:BRSET DEST,#-,waitDEL   ; wait for SDEL being detected on any one link.
        MV R0,AVLN              ; MOVED.
        MV SELDST,AVLN          ; MOVED.
        ANI CR,#----            ; reset timer.
        XOR R0,AVLN,USLN        ;
        SRLSLN @R0              ;
        MV GP1,LNTHWR           ; length received.
        MVI GP0,#----           ; address in shared memory where data is to be
                                  stored.
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP2,#----           ; control info.
        NOP                     ; The information learned during learning mode
        ST [GP0+#0],GP2         ; routing is available in receive FIFO.
        ST [GP0+#1],SADD        ; Move them to the shared memory at the
        ST [GP0+#2],DADD        ; address specified by GP0.
        ST [GP0+#3],GP1         ;
        ADD GP0,#4              ;
        RFIFO GP2               ;
        MV GP7,GP2              ; save it as an immediate add. to destination.
        ST [GP0+#0],GP2         ; This will be useful to update link info.
        BEQZ GP1,cont           ;
        SBI GP1,#1              ; MOVED.
        ADI GP0,#1              ; MOVED.
moredta:RFIFO GP2               ;
        ST [GP0+#0],GP2         ;
        BNEZ GP1,moredta        ;
        SBI GP1,#1              ; MOVED.
        ADI GP0,#1              ; MOVED.
cont:   ANI CR,#----            ; release the bus.
        ORI ICR,#----           ; send interrupt to TR.
        MVI CINFO,#----         ; control for CACK.
        ORI SRC,#----           ; start Tx.
        MV ENCODE,USLN          ; Update the table.
        ENC GP2                 ;
        MV GP0,TABLE            ;
        SHR GP3,DLR,#-          ;
        SBB GP2,GP3,GP2         ;
        ADD GP0,GP0,GP2         ;
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        NOP                     ;
        NOP                     ;
        ST [GP0+#0],GP7         ;
        JMP setwind             ;
        ANI CR,#----            ; MOVED.

/* fault routing initiation at source node. */
startFT:ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        ORI SR2,#----           ; set the type of Tx.
        ORI SR1,#----           ; and int. serviced.
        LDI DADD,[GP0]          ; destination address.
        LDI GP1,[GP0]           ; immediate adds. for message.
        SHR GP3,DLR,#-          ; get the number of links.
        ANI GP3,#----           ; mask the MSBs.
        MV GP0,TABLE            ; address where the links corres. to adds. are
                                  stored.
find:   LDI GP4,[GP0]           ; get the first address.
        STEQ GP5,GP4,GP1        ; compare the addresses.
        BNEZ GP5,found          ; It is required to find the link which
        NOP                     ; corresponds to the given address.
        NOP                     ; Search for it in the table.
        SBI GP3,#0001h          ; decrement the count.
        BNEZ GP3,find           ;
        NOP                     ;
        NOP                     ;
found:  MV DECODE,GP3           ;
        DEC GP3                 ;
        AND GP5,GP3,USLN        ; check if the found link is busy.
        BNEZ GP5,linkbsy        ;
        NOP                     ;
        NOP                     ;
        OR AVLN,AVLN,GP3        ; set the link.
        MV R0,AVLN              ;
        MV USLN,AVLN            ;
        LDSWR @R0               ; load switch register.
        MVI CINFO,#----         ; Send the message on the set link.
        ORI SRC,#----           ;
        ORI CR,#----            ;
waitack:BRSET DEST,#-,waitack   ; and wait for CACK.
        NOP                     ;
        NOP                     ;
        SRSTLN                  ; Reset the link, once the message is
        MVI SELDST,#0000h       ; received.
        JMP setwind             ;
        ANI CR,#----            ; MOVED.
linkbsy:ORI CR,#----            ; send BREQ.
wait:   BSET SR1,#-,wait        ;
        MVI GP0,#----           ; shared memory address for TR.
        MVI GP1,#----           ; control info.
        ST [GP0+#0],GP1         ;
```

```
        ST [GP0+#1],DEST        ;
        ST [GP0+#2],SADD        ;
        ORI ICR,#----           ; send interrupt to TR.
        JMP setwind             ;
        ANI CR,#----            ; release the bus. MOVED.
/* Fault routing at intermediate and destination nodes.*/
FT:     ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        ORI SR2,#----           ; set the type of Tx.
        ORI SR1,#----           ; and int. serviced.
        LDI DADD,[GP0]          ; destination address.
        STEQ GP1,SADD,DADD      ;
        BNEZ GP1,destFT         ;
        NOP                     ;
        NOP                     ;
        LDI GP1,[GP0]           ; immediate adds. for message.
        LDI GP2,[GP0]           ; immediate adds. for cack.
        SHR GP3,DLR,#-          ; get the number of links.
        ANI GP3,#----           ; mask the MSBs.
        MVI GP7,#0002h          ; we need to compare two addresses.
        MV GP0,TABLE            ; address where the adds. corresponding to links
                                  are stored.
find:   LDI GP4,[GP0]           ; get the first address.
        STEQ GP5,GP4,GP1        ; compare the addresses.
        BNEZ GP5,found          ; Search for addresses in the table for
        NOP                     ; corresponding links.
        NOP                     ;
        STEQ GP5,GP4,GP2        ;
        BNEZ GP5,found          ;
        NOP                     ;
        NOP                     ;
        SBI GP3,#1              ; decrement the count.
        BNEZ GP3,find           ;
        NOP                     ;
        NOP                     ;
found:  SBI GP7,#0001h          ;
        MV DECODE,GP3           ;
        DEC GP3                 ;
        AND GP5,GP3,USLN        ; check if the found link is busy.
        BNEZ GP5,linkbsy        ;
        NOP                     ;
        NOP                     ;
        BEQZ GP7,cont           ;
        OR AVLN,AVLN,GP3        ; set the link.
        NOP                     ;
        JMP find                ; find another link.
        NOP                     ;
cont:   MV RO,AVLN              ;
        JMP setwind             ;
        LDSWR @RO               ; load switch register.(MOVED)
/* at the destination .. */     ;
destFT: LDI DADD,[GP0]          ;
        LDI GP2,[GP0]           ; immediate address for CACK.
        SHR GP3,DLR,#-          ;
        ANI GP3,#----           ; mask the MSBs.
        MV GP0,TABLE            ; address where the adds. corresponding to links
                                  are stored.
find:   LDI GP4,[GP0]           ; get the first address.
        STEQ GP5,GP4,GP2        ; compare the addresses.
        BNEZ GP5,found          ;
        NOP                     ;
        NOP                     ;
        SBI GP3,#0001h          ; decrement the count.
        BNEZ GP3,find           ;
        NOP                     ;
        NOP                     ;
found:  MV DECODE,GP3           ;
        DEC GP3                 ;
        AND GP5,GP3,USLN        ; check if the found link is busy.
        BNEZ GP5,linkbsy        ;
        NOP                     ;
        NOP                     ;
        OR AVLN,AVLN,GP3        ; set the link.
        MV RO,AVLN              ;
        MV USLN,AVLN            ;
        LDSWR @RO               ; load switch register.
waitEDE:BRSET DEST,#-,waitEDE   ; wait for EDEL.
        NOP                     ;
        NOP                     ;
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP1,#----           ; control code for TR.
        MVI GP0,#----           ;
        ST [GP0+#0],GP1         ;
        ORI ICR,#----           ; send interrupt to TR.
        MVI CINFO,#----         ;
        ORI SRC,#----           ; start Tx. -- CACK.
        JMP setwind             ;
        SRSTLN                  ; reset the links.
/*      at intermediate nodes due to path grant message on the ring. This is
        executed at nodes which are not in the path. */
cackTR: MV RO,AVLN              ;
        RLSLN                   ; release all links in case of other nodes.
        JMP setwind             ;
        RSSQ @R0                ; (MOVED)

/* On EDEL interrupt.. message received at destination.*/
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,#----           ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
        BRSET DEST,#-,ahead     ; go to ahead if this is not the last frame.
        SBB GP2,LNTHWR,LNTHRD   ; find the size remaining to be written. (MOVED)
        MOV LNTHRD,LNTHWR       ; (MOVED).
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ; wait for the bus.
        MVI GP1,#----           ; send path release message
        MVI GP0,#----           ; address in the shared memory.
        ST [GP0+#0],GP1         ;
        ANI CR,#----            ; release the bus.
        ORI ICR,#----           ; send int. to TR.
        MVI CINFO,#----         ; DACK frame.
        ORI SRC,#----           ; start TX.
        MV RO,USLN              ;
        MV SELDST,USLN          ; reset selector to destination.
        SRSTLN                  ; reset links.
ahead:  ORI ICR,#----           ; send Rx to host.
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP1,#----           ; DMAC add.
        MVI GP0,#----           ; add. of the FIFO.
        ST [GP1+#0],GP0         ;
        ST [GP2+#0],GP0         ; length to be transferred.
        ORI CR,#----            ; send DREQ.
waitTC: BRSET SR1,#-,waitTC     ; wait for TC to be received.
        NOP                     ;
```

```
        NOP                     ;
        JMP setwind             ; switch the register bank.
        ANI CR,#----            ; release the bus. (MOVED)

/*On CACK interrupt at intermediate nodes */
        BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,#----           ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
        BSET SR2,#-,FRcack      ; is it due to flood routing...?
        BSET SR2,#-,LMcack      ; or learning mode ?
        NOP                     ; or fault routing ?
        NOP                     ;
        MV R0,USLN              ;
        RSSQ,@R0                ; reset states and switch register.
        RLSLN                   ; reset all links used in current Tx.
        JMP setwind             ;
        ANI CR,#----            ; MOVED.
FRcack: XOR R0,AVLN,USLN        ;
        RSSQ,@R0                ; reset states and switch register.
        RLSLN                   ; release links not used.
        MV R0,CKR               ;
        STRST,@R0,#----         ; set it to state = 4 for DACK.
        MV INIT,R0              ;
        JMP setwind             ;
        ANI CR,#----            ; MOVED.
LMcack: MV R0,USLN              ;
        RSST,@R0                ; reset states and counter.
        RLSLN                   ; reset all links used in current Tx.
        ANI CR,#----            ;
        XOR ENCODE,USLN,CKR     ; store the links required to be updated.
        ENC UPDT                ;
        MV ENCODE,CKR           ; link on which msg. was received.
        ENC GP7                 ;
        SHL GP7,GP7,#-          ;
        JMP setwind             ;
        ORI UPDT,UPDT,GP7       ; the link on which CACK was received. (MOVED)

/* on DACK interrupt for Flood Routing. */
        MV R0,USLN              ;
        RSST,@R0                ; reset states and counter.
        RET                     ;
        RSTLN                   ; reset all links used in current Tx. (MOVED)

/* On timer interrupt. error has occured.. either message was not delivered
   or acknowledgement was not received */

error:  BRSET ICR,#-,zero       ; check if priority =0 => go to zero.
        ORI SR1,#----           ; set the register window.
        NOP                     ;
        JMP ahead               ;
        NOP                     ;
zero:   ORI SR1,#----           ; set the other window.
        MV R0,AVLN              ;
        RSTLN                   ;
        MVI GP1,#----           ; check the source bit in SRC.
        AND GP1,SRC,GP1         ; determine if it's source or dest. or
                                  intermediate node.
        BNEZ GP1,srcdest        ;
        NOP                     ;
        NOP                     ;
        JMP ahead               ; reset the links and states at intermediate
        RSST @R0                ; nodes.  MOVED.
srcdest:MVI SELDST,#0000h       ;
ahead:  ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP0,#----           ; shared memory address.
        MVI GP1,#----           ; control info.
        ST [GP0+#0],GP1         ;
        ST [GP0+#1],DEST        ;
        ST [GP0+#2],SADD        ;
        ORI ICR,#----           ; send interrupt to DP.
        MVI GP0,#----           ; shared memory address for TR.
        MVI GP1,#----           ; control info.
        ST [GP0+#0],GP1         ;
        ST [GP0+#1],DEST        ;
        ST [GP0+#2],SADD        ;
        ORI ICR,#----           ; send interrupt to TR.
        JMP setwind             ;
        ANI CR,#----            ; release the bus. (MOVED)

/* on interrupt from data processor. This is for updating the link
   information obtained during learning mode routing. */
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        MVI GP0,#----           ;
        NOP                     ;
        LDI GP1,[GP0]           ;
        BSET GP1,#-,LMupdt      ;
        NOP                     ;
        NOP                     ;
        /* other functionalities not defined */
LMupdt: LDI GP1,[GP0]           ; the node address from which msg. was received.
        LDI GP2,[GP0]           ; the node address on which CACK was received.
        ANI CR,#----            ; release the bus.
        MVI GP3,#----           ;
        AND GP3,UPDT,GP3        ; mask the MSBs
        MV DECODE,GP3           ; determine the links to be updated.
        DEC GP4                 ;
        MVI GP3,#----           ;
        AND GP3,UPDT,GP3        ; get the cack link.
        MV DECODE,GP3           ;
        DEC GP5                 ;
        MV GP0,TABLE            ; find its address in the memory.
        SHR GP6,DLR,#-          ; get the number of links
        SBB GP4,GP6,GP4         ;
        SBB GP5,GP6,GP5         ;
        ADD GP0,GP0,GP4         ;
        ORI CR,#----            ; send BREQ
wait:   BSET SR1,#-,wait        ;
        NOP                     ;
        NOP                     ;
        ST [GP0+#0],GP1         ; store the immediate node address corresponding
        MV GP0,TABLE            ; to the link.
        ADD GP0,GP0,GP5         ;
        JMP setwind             ;
        ST [GP0+#0],GP2         ; (MOVED)

/* end of protocol implementation for data link interface */
```