VISTAnet Network Interface Unit: Prototype System Specification

(TR91-017)

Raj K. Singh
Stephen G. Tell
David Becker

Microelectronic Systems Laboratory
The University of North Carolina
Department of Computer Science
CB# 3175 Sitterson Hall
Chapel Hill, NC 27599-3175

The work undertaken for the VISTAnet project is supported, in part, by the National Science Foundation and the Defense Advanced Research Projects Agency under cooperative agreement NCR 8919038 with the Corporation for National Research Initiatives. Support is also provided by BellSouth Services and GTE Corp.

CONTENTS

VISTAnet Network Interface Unit: Prototype System Specification

1. INTRODUCTION

Emerging trends in high-speed networking and high-performance computing systems have revitalized the research in distributed parallel computing. The notion of metacomputers is being developed, where a large number of heterogeneous computing resources are networked together and operate in tandem to solve problems beyond the scope of the resources of any one system in the cluster. The advent of high-speed networks of increased bandwidth is putting demands on performance and functionality of the host interfaces.

VISTAnet is one of five national research testbeds organized by the Corporation for National Research Initiative (CNRI) and funded in part by NSF, DARPA, and private corporations such as BellSouth and GTE. The key goal of the VISTAnet project is to harness the capabilities offered by emerging public network standards such as ATM and SONET to support interactive 3-D radiation treatment planning by linking three supercomputers namely, Cray Y-MP (Y), MasPar MP-1 (MP), and Pixel-Planes 5 (**PXPL5**) and a Medical Workstation (**MWS**) through a gigabit **BISDN** network.

The Network Interface Unit (NIU) is an intelligent custom communication interface that provides full-duplex I/O channel directly to the message passing token ring network of the PXPL5 machine. It uses a standard PXPL5 ring board providing two pairs of transmit and receive ports to the ring backplane. The NIU interfaces to the ATM/SONET fiber-optic network over a pair of 800 Mbits/s simplex High Performance Parallel Interface (**HIPPI**) links through the Network Terminal Adapter (NTA). The HIPPI interface will be implemented as two simplex channels: one receive (RX) and one transmit (TX).

The general characteristics of the application, computing resources and various interfaces in the network are described in section 2. The constraints imposed due to these characteristics on NIU design will also be described. Section 3 describes an architecture and outlines a proposed implementation. Various hardware and software trade-offs and their impact on future research will also be discussed in this section.

2. CHARACTERISTICS AND CONSTRAINTS

2.1 The VISTAnet Application

The driving application of the VISTAnet project is radiation treatment planning. It is a visually oriented interactive application and generates traffic that requires a low latency but tolerates an imperfect network. The interactive nature of the application requires that the NIU offers a low latency and high throughput path to PXPL5. The visual nature of the data makes it relatively insensitive to the infrequent errors expected on a high-quality fiber-optic network.

2.2 Computing Resources

Preliminary studies show that various computing resources in the network have unique communication requirements. For example, PXPL5 is a multiprocessor system with four subsystem types which communicate via the custom ring network. A brief description of different computing resources is given below.

2.2.1 Cray Y-MP

The Cray Y-MP 8/432 with four processors provides the basic parallel and pipeline computing resource and serves as the workhorse for the numerical computation associated with radiation dose calculation and visualization. For communication with other resources, we will use the high-speed HIPPI interface and its associated drivers.

2.2.2 Medical workstation

The Medical Workstation (MWS) serves as a front end for user interaction with the application. In the minimum configuration, it will provide a display monitor for generated images and some input devices such as joystick, keyboard, and mouse.

Although the final MWS system configuration is yet to be determined, we will explore three proposed hardware configurations and assess their impact on design and development of network interfaces, upper-level protocol and related driver software. Our intent is to design the NIU to support any of the following three proposed configurations for the MWS.

1. A commercial workstation with built-in high-resolution framebuffer. The workstation would be connected to the ATM/SONET network through a smart HIPPI interface card with a master interface for the native bus of the host workstation. This HIPPI interface card would be connected to an NTA. This configuration would offer an advantage in interesting network related research by allowing small control packets from pointing devices to be mixed with large data packets on the same high-bandwidth network.

2. A scaled-down PXPL5 system with minimal resources. For example, a one-rack system with communication ring, few renderers, few GPs and a host-interface. We will refer to this system as a mini-PXPL. The host for the mini-PXPL would have to be a Sun-4 workstation. This arrangement would also support generation of mixed traffic by channeling the control data packets into the network through NIU. It will also support research into additional ways to decompose the problem among distributed computing resources. As we will see in the next section and later, a network with multiple PXPL5 machines poses some unique problems.

3. A commercial framebuffer with on-board HIPPI interface and a monitor. The framebuffer acts as a dumb peripheral device and connects to an NTA. Since a dumb framebuffer is not directly coupled with the workstation, the MWS would interact with other nodes through Internet or dial-up connections.

2.2.3 Pixel-Planes 5 and its salient features

The primary application of the Pixel-Planes 5 (**PXPL5**) system is real-time rendering of 3-D graphics images. The PXPL5 system is a collection of heterogeneous computing resources connected through a high-speed ring network. The three primary resources are: (1) Graphics Processor (GP), (2) Renderer, and (3) Framebuffer. GPs process 3-D polygon data to generate commands for the Renderers. These commands cause the Renderers to create an image that is sent to the Framebuffer for display. A fourth resource, the host interface (HIF), provides the communication to the host workstation.The PXPL host workstation is a SUN-4 system which is required for initialization and monitoring of PXPL5, but is not involved in the main data flow of the application. The remaining components of the PXPL5 system require more detailed examination as they are used heavily by the application.

2.2.3.1 Ring Network: Each ring port (node) provides both receive and transmit channels at a sustained transfer rate of 20MW/sec (i.e. 32-bit words at 640 Mbps). Internally the ring is implemented with a high-speed clock (160 MHz or 6.25 ns period). The channel acquisition time is proportional to the total number of ports on the ring. Since there can at most be 128 ports on the ring, the maximum ring latency is equal to (128*6.25 ns*2.5 = 2 μs). The factor of 2.5 accounts for two ring cycles, one each for acquiring a channel and for acquiring the receiver and some additional overhead. Actual channel acquisition delay can be arbitrarily long as application boards will not allow their respective receivers to be acquired until their input buffers are sufficiently empty. There is no provision for polling of receivers; once a transmitter begins sending a message, it is blocked until the addressed receiver is acquired and the message delivered.

The ring boards provide a message passing system between the ring ports on each application board. The beginning and the end of a message are delimited by out of band hardware signalling; size is typically included in the message. The general form of all ring messages is shown in Fig. 1(a). The first word of each message, called the Ring Address Word (**RAW**), contains a 7-bit **Ring Node Address** that is interpreted by the ring boards as the destination port of the message. The remaining 25 bits are encoded as per application software requirements.

The following sections examine message formats used by each of the existing application boards. Based on these message formats, we propose an addressing mechanism for integrating the NIU in the ring and the network.

2.2.3.2 Graphics Processor: GPs can send messages as large as their memory. However, the size of incoming messages is limited to 1792 words. The message format of the GP is completely under control of software running on the Intel i860 microprocessor on board. This software, known as the Ring Operating System (**ROS**) uses the message format shown in Fig. 1(d).

Mailboxes are a software abstraction provided by ROS for use by upper layer software. The mailbox number is in effect another level of addressing that identifies different tasks in the upper layer software such as the **PPhigs** polygon graphics system, or the volume rendering system.

2.2.3.3 Frame Buffer: The two ports on the frame buffer are used for different types of messages; each message contains only one command. Both ports use the message format shown in Fig. 1(b). Each command, however, must be sent to a particular port. Commands with display data are sent to port 0 or 1 depending on the target memory bank. These commands use the 18-bit memory address field of the address word. The commands for writing the color look-up table, cursor position, or cursor shape include a variable number of additional words and ignore the memory address field of the address word. The **Write CSR** and **Write Address Register** commands are each followed by a single 32-bit word. The **Read CSR** command consists only of the address word. The status is sent to the ring address set by the most recent **Write Address Register** command. The following table summarizes the format of all six framebuffer commands. The "length" column refers to the total length of the message including the address word and any data words.

Command	**Length**	**Interpretation of address data words**	
Write Data	2≤L≤N	Address word:	18 bits of memory address 3 bits of command code
		Data Words:	Display data (N is a multiple of 128)
Write Ramdacs	2≤L≤N	Address word:	command code only
		Data words:	Ramdac address and data (N is a multiple of 2)
Write Cursor	2≤L≤N	Address word:	command code only
		Data words:	Cursor chip address and data (N is a multiple of 2)
Write CSR	2	Address word:	command code only
		Data word:	5 bits of new CSR value
Read CSR	1	Address word:	command code only

Write Address Register	2	Address word: Data word:	command code only New value for address register
Read Address Register	2	Address word: Data word:	command code only return address to which register value is sent.

2.2.3.4 Renderer: Each renderer application board contains two ports to the ring. One port is labelled **IGC Port** and the data received at this port is interpreted by the microcode sequencer called Image Generation Controller (IGC). The IGC generates instructions and control for the array of custom SIMD processors.

The other ring port is known as the backing store port. The backing store (**BS**) is composed of dual-ported video RAMs (**VRAMs**) with one port connected to the SIMD array and the other to the BS port. The memory system is fast enough to provide a path from the backing store to the ring at a sustained rate of 20MW/sec. The SIMD array can be instructed through the IGC to write data into the backing store. Once the data is in the backing store, it can be transferred to other devices on the ring.

The Renderer is a typical dumb device; transfer of data to and from backing store must be coordinated by other devices on the ring. In order to transfer data out of backing store, some ring device must send a command to the backing store port instructing it to send data to a particular ring address. The ring device sending the **BS_TRANSMIT** command (Fig. 1(c), Opcode=0x3) to the backing store port need not be the one receiving the data. The **BS_TRANSMIT** command is followed by a destination address word which is used verbatim as the RAW for the message containing transmitted data. This aspect of the renderer backing store operation constrains data flow from the renderer to the NIU (Ref: PXPL5 Renderer Functional Description Sec. III.4.1.-2) .

The backing store port command format is shown in Fig. 1(c). Multiple commands can be sent in a single message. These commands are of varying length. The first word of a command contains a 4-bit opcode. If the opcode is a no-op, the remaining bits of the command word are ignored; for other opcodes all 25 application-board-specific bits are significant and have the meanings as shown in the Fig. 1(c).

2.2.4 Classification of Resources

The computing resources, described in previous sections, can be categorized based on their communication capabilities. It is important to distinguish between two classes of devices: **smart** and **dumb** devices. A smart device is one that is capable of processing arbitrarily formatted data. Such devices can be programmed in software to handle changing protocol requirements and support flow control. Example of such devices are Y, GP, MP and MWS. A dumb device, on the other hand, is one that is constrained to fixed data formats since all processing is implemented in hardware. It provides limited capability for flow control and error handling. Examples of dumb devices are Renderer and Framebuffer. This poses significant constraint on the design of NIU.

The following section describes the network characteristics across the HIPPI network. The implementation section will later return to the details of messages and addressing styles among application boards and their role in the network.

2.3 The Network

A network is generally characterized by its hardware and software components. At the lowest level, networks provide an unreliable packet delivery due to lost or destroyed packets due to hardware failure and delivery of packets in scrambled order. To provide a reliable packet delivery system, a stack of upper layer protocols (ULPs) with the capability for positive acknowledgement and retransmission is implemented on top of the hardware layer. In the following section, we estimate the NIU requirements from the standpoint of supporting a protocol stack that will enable a reliable delivery of packets.

31 25	24 0
Ring Node Addr (7 bits)	Application Board Specific (25 bits)
DataWord*	

Fig.1(a): Generic Ring Message Format

31 25	24 21	20 3	2 0
Ring Node Addr (7 bits)	unused (4 bits)	Mem Addr (18 bits)	Cmd (3)
DataWord*			

**Fig.1(b): Framebuffer Message Format.
Mem Addr used for certain commands only.**

31 25	24 18	17 11	10 4	3 0
Ring Node Addr (7 bits)	# lines (7 bits)	sector # (7 bits)	start line (7 bits)	Opcode (4)
Data Word *				
	# lines (7 bits)	sector # (7 bits)	start line (7 bits)	Opcode (4)
Data Word *				

**Fig.1(c): Renderer Backing Store Message Format.
Multiple commands may be sent in same message.**

31 25	24 12	11 0
Ring Node Addr (7-bits)	Msg Size in Words (13-bit)	Mbox # (12-bits)

Fig. 1(d): Ring Operating System (ROS) Message Format

2.3.1 Performance: overhead, throughput and flow control

The hardware complexity of various NIU subsystems is directly dependent on the division of tasks (between software and hardware), network buffering capacity, and flow-control mechanism (acknowledgement, retransmit etc.) supported by the upper layer protocol (**ULP**).

The following paragraph describes the basic network components such as SONET and ATM protocol and discusses their impact on the design of protocol processing subsystem of NIU. This will result in specifications for the size and speed of various building blocks of the protocol processing subsystem. It must, however, be noted that certain constraints are imposed due to the state-of-the-art in technology and availability of off-the-shelf components. For example, we have decided to limit the HIPPI packet size to 16KB due to the size limitation of available fast FIFO devices. In future, as bigger and faster components become available, the older devices can be replaced to update the hardware system. A simple software change at different hosts will enable a full access to the enhanced capabilities of the NIU. The basic network characteristics are as follows:

Maximum SONET throughput	= 622 Mbps * 0.9654 (overhead) ≈ 600 Mbps
Maximum ATM cell payload size	= 44 Bytes
Total ATM cell size with header	= 53 Bytes (5B header + 4B adaptation)
Maximum ATM throughput	= (44/53) * 600 Mbps ≈ 498 Mbps
Maximum size of HIPPI packet	= 16 KB (4KW)
Maximum size of HIPPI burst	= 1 KB (256W)

Thus, with the maximum bandwidth of 498 Mbps between NTA and NIU

HIPPI burst service time	= (1KB * 8)/498 Mbps ≈ 16.5μs
HIPPI packet service time	= (16KB*8)/498 Mbps ≈ 263μs

A PXPL5 ring port can transfer data at the peak rate of 640 Mbps. However, the effective throughput is slightly lower due to added overhead shown below:

Ring channel acquisition time	= 3μs
Ring message payload (4KW)	= (4KW*32)/640Mbps = 204μs
Start and end of message overhead	≈ 1μs
Effective HIPPI packet service time	≈ 208μs.

The time difference between a packet received from the network and the time required to transmit it over the ring is (263-208)μs = 55μs. This time is available for the time-critical ULP related processing, i.e., acknowledgement, retransmit, and local control processing. In the best case estimate, with a high performance processor (e.g. 25MHz RISC processor with 40ns period and average of 1.5 cycles/instr), one can execute nearly 55μs/(40ns*1.5) ≈ 916 machine instructions in this duration.

To determine the size of on-board memory, we need to estimate the buffering capacity of the network. We estimate the maximum length of the fiber between MCNC and UNC as 20 miles (or 32 km). With the speed of light in fiber at 200,000 km/s, the time of flight ~ 160 μs.

HIPPI burst data in fiber	= 160μs * 1KB/16μs ≈ 10 KB
Buffering capacity of ATM network	= 1400 cells * 44B/1024 ≈ 60KB
Buffering capacity of NTA	= 32 KB

Thus, there is nearly 102KB (~6.4 HIPPI packets) of data in transit in one direction, or a total of 13 HIPPI packets in flight in both directions. The buffer size of 64KW or 16 HIPPI packets should be sufficient to support the flow control algorithm of ULPs. These calculations indicate that we can experiment with a wide range of window sizes in software.

Excluding the adaptive time-out capability of many protocols, to establish a time-out period for retransmission from the NIU, we need to estimate the worst case turn-around time for the acknowledgement. This is directly dependent on the number of packets in the network and the packet processing overhead at the Y. In the worst case scenario, with 13 packets in the network in both directions (13*254µs) and about 10µs packet processing overhead at the Y, the time-out can be set in the vicinity of 3-4 ms.

2.3.2 Protocols and Interfaces

The NIU's primary role is to provide a high speed data transfer channel between PXPL5 and remote hosts. In order to maximize the data throughput, we have decided to minimize the protocol overhead by trimming the protocol stack. This decision is also in concurrence with the requirements of low latency and high error tolerance of the underlying application in our network.

The deliverable NIU will implement the following stack of protocols:

HIPPI-PH	Transfer of bursts of data to the NTA
HIPPI-FP	Unreliable connectionless datagram service
	(Optional standard and experimental protocol layers)
RingP	Encapsulated PXPL5 ring message delivery service

Nevertheless, NIU will provide a flexible platform that will support many different protocols through changes in software. It will provide for protocols such as TCP/IP, XTP, TP4, and others running at gigabit rates by performing common computation-intensive tasks, such as checksum computation and data transfer, in hardware. Future research into network protocols will involve changing the software running in the NIU's microprocessor. The HIPPI-PH, HIPPI-FP, and RingP layers are required by the NIU hardware design. The following sections describe each of these three layers at a behavioral level.

2.3.2.1 HIPPI-PH: The HIPPI-PH protocol provides for physical signaling to transfer 256-word bursts of data and to delimit packets composed of these bursts. It provides the test for transmission related errors by employing a 2-D parity and LLRC checks on each burst. However, certain types of error may still go undetected. It does not provide any mechanism for error correction and recovery. The HIPPI-PH is based on two simplex connection-oriented channels. A 32-bit **I-field** is used at the start of a connection to convey addressing, routing and control information. The HIPPI-PH data format is shown in Fig. 2.

2.3.2.2 HIPPI-FP: The HIPPI-FP provides an unreliable datagram service to its client upper layer protocols. It supports multiple protocols through the use of **ULP-ID** byte in its header. It breaks and establishes HIPPI-PH connections, as required, based on the destination of each datagram. The FP layer breaks up client protocol data (**data set**) into HIPPI-PH bursts. HIPPI-PH packets transmitted by HIPPI-FP consist of three areas, called the Header, D1 area, and D2 area, each starting and ending on a 64-bit boundary. Figure 3 shows the HIPPI-FP packet format.

Each client datagram consists of two parts, a short D1_data_set and a long D2_data_set. However, each of these may be omitted from the FP-packet. A D1 and D2 data set pair is sent in its own HIPPI-PH packet. The size of the D1 and D2 data sets is delivered, although the D2 size can be left unspecified at the start of the packet by setting the D2_data_size to a value of 0xffffffff.

This unspecified D2 size feature will not be supported by the NIU. The use of HIPPI-FP is desirable as it allows for interoperability among testbeds and emerging computing systems with standard HIPPI interfaces.

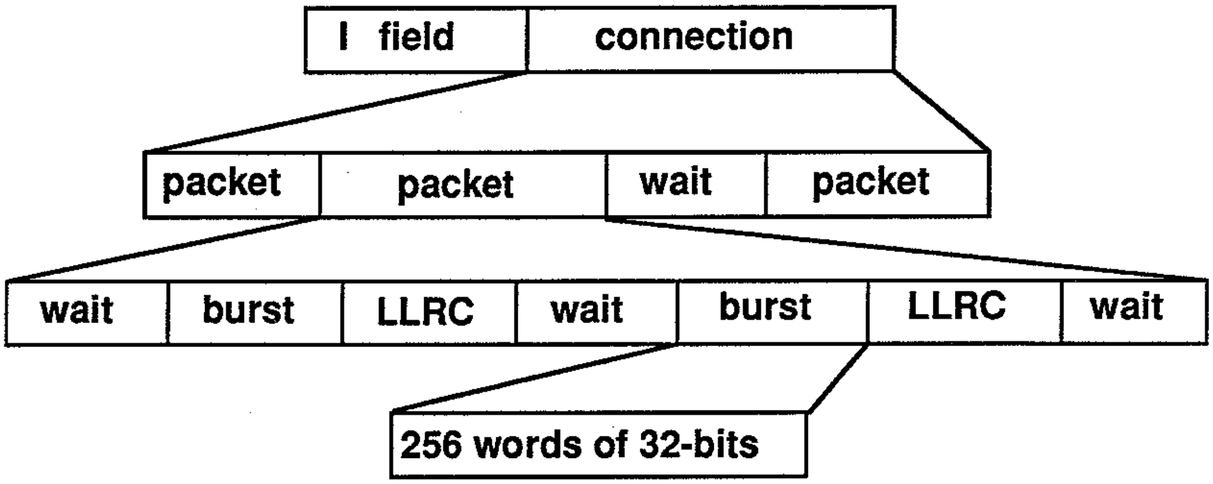

Fig.2: HiPPI-PH Data Format

The processing of the HIPPI-FP header will be carried out by the NIU's microprocessor. The intermediate protocol layers are at liberty to use the D1 and D2 data sets and ULP-ID as desired. Multiple intermediate protocols can be supported through the use of different ULP-IDs.

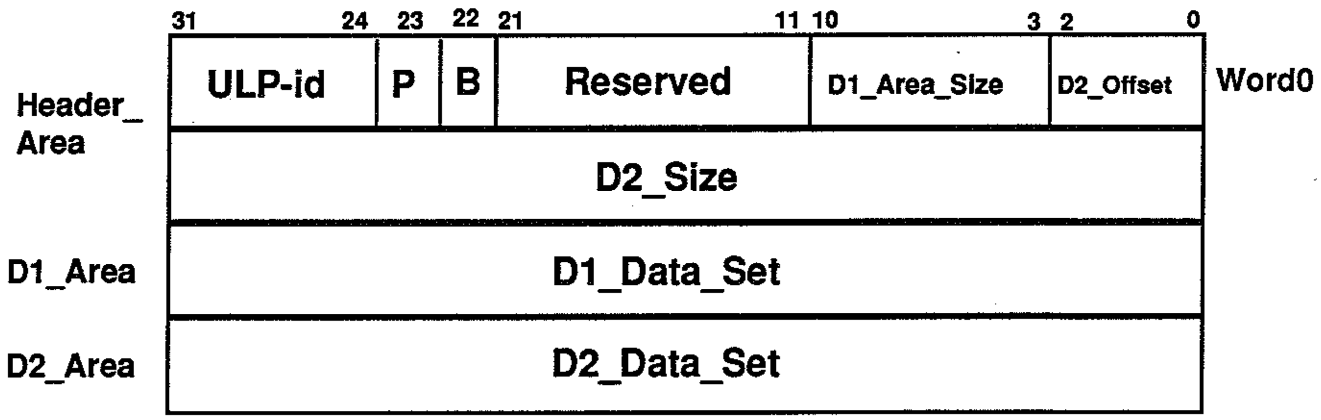

Fig. 3: HiPPI-FP Packet Format

2.3.2.3 Intermediate Protocols: To allow maximum flexibility in future research, the NIU makes minimal assumptions about the protocol layers located between HIPPI-FP and RingP. The protocol software can determine the number of header words required by the intermediate protocols. A 16-bit 1's complement checksum is computed over the entire arriving HIPPI packet. The software can determine how this checksum relates to any headers in the intermediate layers.

The intermediate layers can be omitted completely. A simple control program can pass data directly to the RingP layer after HIPPI-FP processing is completed. Initially, NIU will be operated in this manner. More complex protocol stacks can be added later. The NIU research document outlines the implementation of a standard protocol stack consisting of the HIPPI-LE, IEEE 802.2/SNAP, IP, and TCP layers in some detail.

For convenience, we will refer to the variable contents of this intermediate layer as To-Be-Determined-Protocol (**TBDP**). It has been proposed that the standard TCP/IP stack will be the first complete protocol stack to be implemented. In this case TBDP can be read as "TCP." The transfer units or packets used by TBDP will be referred to as "TBDP transfer units". In case of TCP/IP, the transfer unit is called a **segment.**

All arriving HIPPI data is treated identically by the associated NIU. When a HIPPI packet arrives at the NIU, the HIPPI-FP and intermediate protocol related data will first be processed as described later. The intermediate protocol's payload will contain RingP-encapsulated ring messages. These messages are placed directly on the ring without further processing.

RingP is a simple layer that encapsulates various forms of PXPL5 ring messages. It consists of a single header word prepended to each ring message indicating its length. The use of a length word provides record boundaries between messages for use with recordless stream protocols such as TCP/IP in intermediate layers. RingP can also be used in a **synchronized** mode with intermediate protocols that provide record boundaries. In synchronized mode each transfer unit of the intermediate protocol contains an integral number of complete ring messages; ring messages do not span intermediate protocol's transfer units.

3. NIU ARCHITECTURE AND IMPLEMENTATION

The NIU will consist of the following three major subsystems:

- HIPPI Interface
- Network Protocol Handler
- Ring Interface

A nomenclature for NIU channels is established to avoid confusion of relative terms like "transmit" and "receive". The channel routing data from the NIU to the outside world is called the **network-bound** channel; the channel routing data from the outside world to the NIU is known as the **ring-bound** channel.

In designing the NIU, attention will be paid to providing adequate testability. Additional data paths will be provided to exercise the protocol processor and data buffers through ring port1.

Figure 4 shows a block diagram sketch of the protocol processor subsystem. It should be emphasized that the design is in development phases. A system level simulation will help in refining details of the specific circuit design.

3.1 Introduction: Hardware/Software decomposition philosophy

To accommodate the high data transfer rate, many NIU functions will be implemented using fast programmable logic devices. Some NIU functions will be implemented with the help of a high-performance microprocessor to provide flexibility for future research. The exact division of tasks between hardware and software has been based on detailed analysis of these requirements.

3.2 HIPPI interface subsystem

3.2.1 HIPPI-PH

The HIPPI interface will consist of two simplex channels; one each for transmit (HIPPI-PH source) and receive (HIPPI-PH destination). The NIU design will be implemented using high-performance, low power TTL and CMOS devices. At the destination port, the differential ECL signals from the HIPPI cables will be converted to TTL signals using 100K series ECL-to-TTL receivers. Similarly, at the source port, the differential ECL signals will be generated using 100K series TTL-to-ECL drivers. The signal levels and timing will conform to the HIPPI-PH standard specifications (Ref: X3T9.3/88-023 REV7.1).

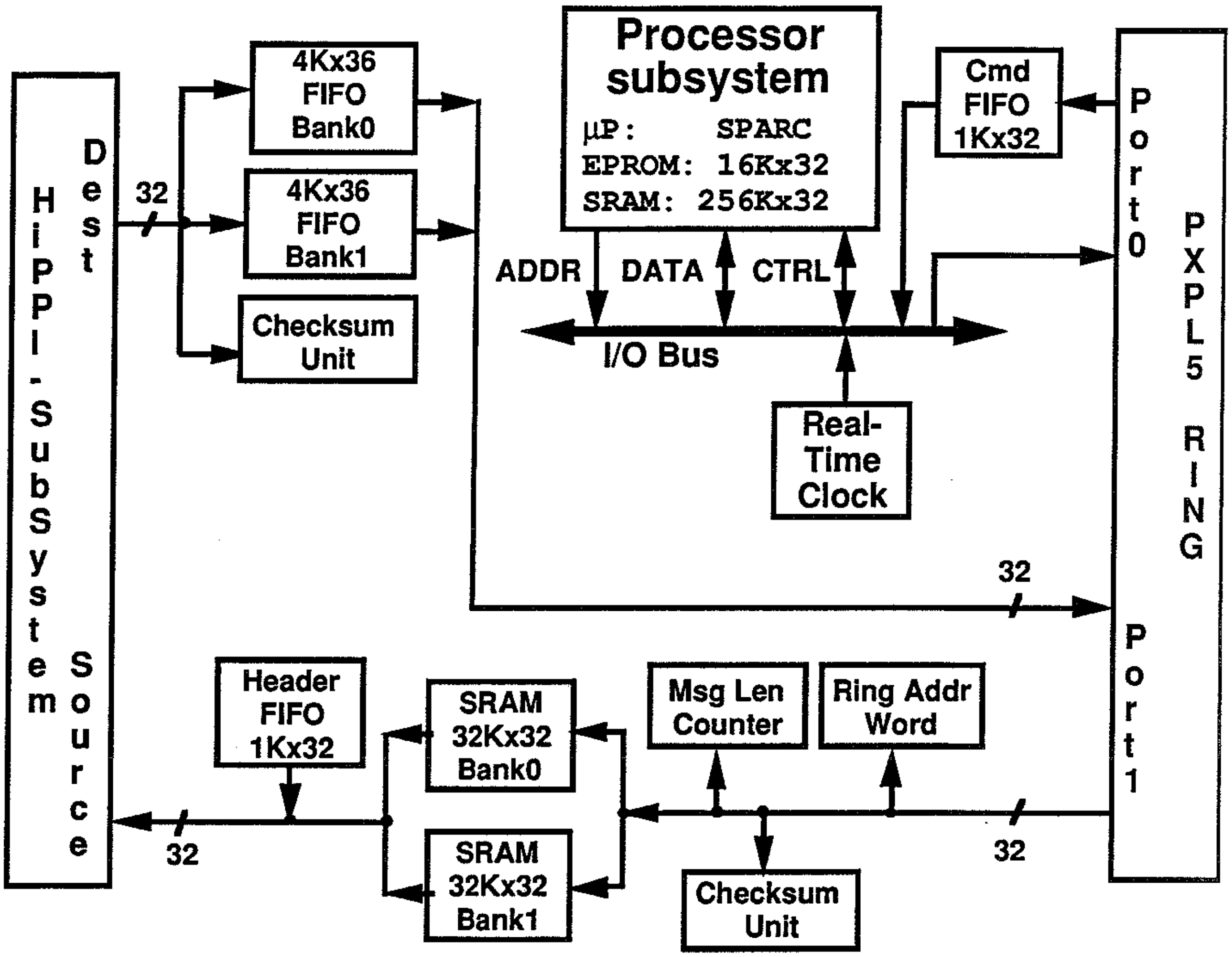

Fig. 4: Protocol Processor Subsystem

3.2.1.1 Connection, LLRC and Parity: Hardware state machines in the HIPPI interface subsystem will handle the connection initiation and termination as per the HIPPI-PH specification. The HIPPI interface subsystem will check the parity and LLRC information on the received data and generate the same for the transmitted data. It will interact directly with the data buffers of the network protocol subsystem to transmit and receive data. While a packet is received and placed into a FIFO buffer, the LLRC and parity evaluation hardware will set a flag if an error is detected anywhere in the packet. This flag will be the primary indication to the protocol software whether a packet was received correctly.

3.2.1.2 Ready pulse generation: The flow control of HIPPI-FP using **ready** pulses for each burst requires special attention because the FIFO buffers of the NIU are packet-oriented instead of burst-oriented. In order to send the correct number of ready pulses to accept a long packet without being overrun by possible small packets, we propose to first send a single ready pulse to receive the first burst of a packet. Later, based on the length in the HIPPI-FP header, we will send the appropriate number of additional ready pulses to receive the remainder of the packet.

Software reading the header will write to a hardware register called the destination ready pulse counter. When written with a nonzero value, the hardware will count down and generate the specified number of ready pulses. Software control of this counter allows other ready pulse strategies to be explored. This counter should not be confused with the count of incoming ready pulses at the HIPPI source port, which is handled entirely in hardware.

3.2.2 HIPPI-FP

The implementation of the HIPPI-FP layer in the NIU will be distributed among various subsystems. The HIPPI interface subsystem hardware will handle the function of segmenting packets into HIPPI-PH bursts for transmission, and reassembling received bursts into contiguous packets. Most of the other HIPPI-FP tasks will be handled by software.

Software in the microprocessor will keep track of the HIPPI I-field addresses used for network routing. When a different I-field is required from the previous packet, the software will signal to the HIPPI subsystem to break the current connection and initiate the new.

The HIPPI-FP offset and fill features will not be supported by NIU. An offset at the start of the D2 area is not required because the RingP messages consist of 32-bit wide words. The fill area at the end of the D2 data set is not supported due to increased complexity of the hardware to discard data at the end of the packet. Generally, after the header words are removed from the front of the packet, the hardware controller will deliver the data to the ring until the end of the packet is reached. The overall result of this implementation is that a short HIPPI burst, if any, must occur at the end of the packet. An instance of such a packet is illustrated in Figure 3b of the Appendix B of HIPPI-FP standards document.

3.3 Ring Interface

The ring interface provides the communication link between the NIU and the PXPL5 system. Each of the two ports, namely port 0 and port 1, provide simultaneous transmit and receive capabilities. The port 0 will be used for communications directly to the microprocessor. It will be designated as the command port and used for system initialization, testing, and monitoring tasks. The port 1 will be used to transfer ring messages to and from the network. It will therefore be designated as the data port.

3.3.1 RingP: all ring-bound communication

Only the lower 16 bits of the 32-bit RingP header word are used to encode the length of the message. A length of one indicates a message consisting only of RAW and no data words. A length of zero indicates that there is no valid ring message. The header with zero length can be beneficially used to pad the HIPPI-FP packet in building a packet containing an even number of 32-bit words. The remainder 16 bits are reserved and should be set to 0 for compatibility with any future work.

A timeout mechanism will be implemented on the data port to recover from the situation where a corrupted RAW selects a nonexistent ring device and the NIU would otherwise hang waiting for the receiver to be acquired.

3.3.2 Message routing and address styles

The application devices in the PXPL5 ring network are assigned a fixed hardware address of 7 bits. In view of the emerging possibility of multiple PXPL5 resources in the same network, we have developed an augmented addressing scheme that extends over the entire ATM/SONET network. The modified address is composed of two parts, namely, **major** or **global** address and **minor** or **local** address.

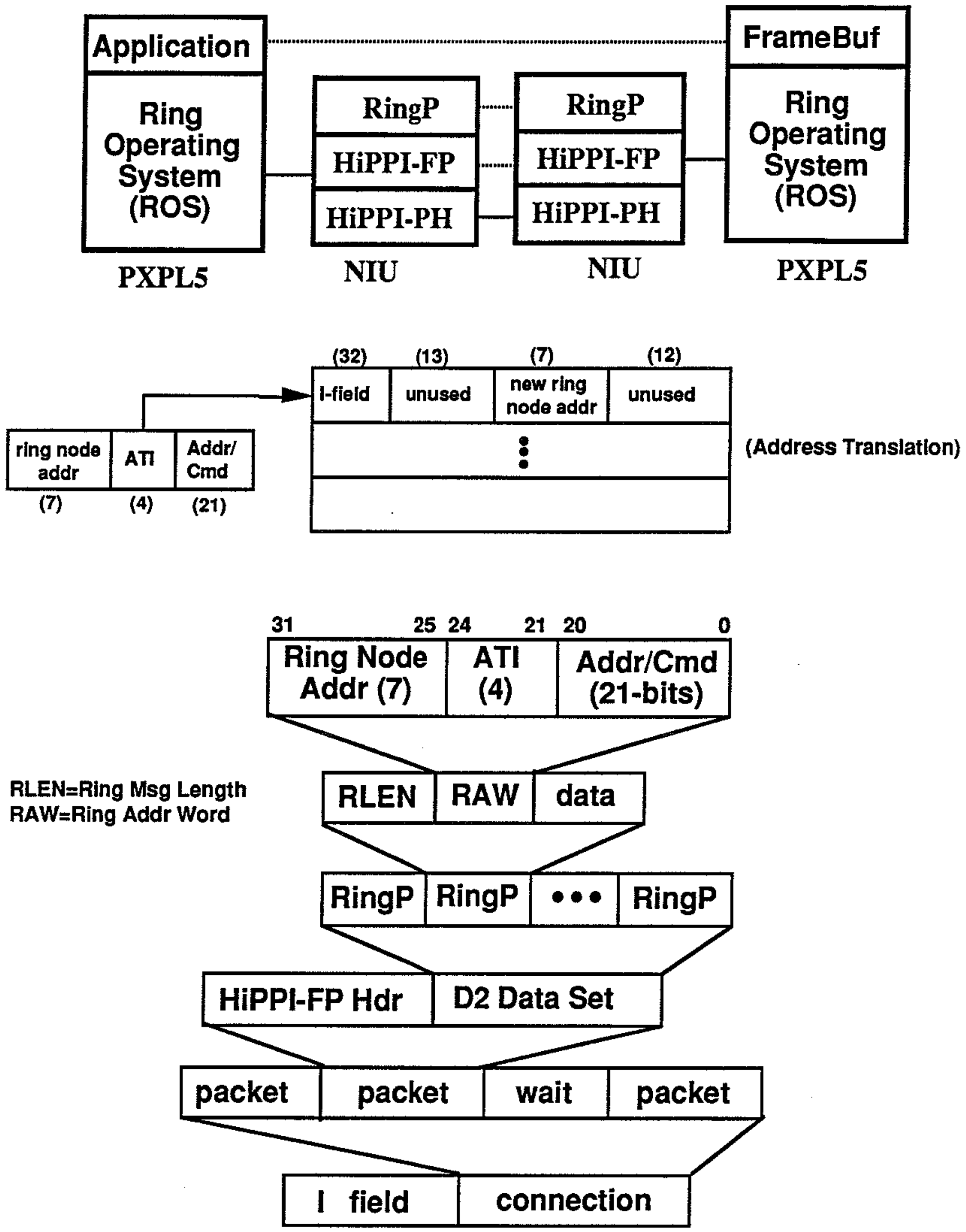

Fig.5: Peripheral Style Communication and Associated Packet Format

A major address is associated with each of the computing resources, Y, MWS, MP, and PXPL in the ATM/SONET network. The minor address serves as the ring node address if the major address identifies a PXPL5 system. However, a PXPL5 system is connected to the network via an NIU. Thus, NIU plays the role of a gateway and would be assigned a major address. It provides the address translation capability through look-up tables to route data to correct destination devices in the network. This address translation is carried out by software in the microprocessor, and can be modified or omitted if necessary.

The derivation of the routing information varies with different source and destination devices. Two unique communication styles with associated addressing requirements have been identified: (1) the **peripheral** style, and (2) the **host** style. The peripheral style addressing is typically used for communication whenever a dumb device is involved, while the host style addressing is primarily used between smart devices. Figures 5 & 6 show the details of peripheral and host style communications, respectively.

3.3.2.1 Peripheral style network-bound communication

The peripheral style communication was developed due to the need for a Renderer to send data through the network to a Framebuffer located on a distant PXPL5 ring. The Renderer is constrained because only one word, namely, the RAW can be prepended to outgoing backing store data. The RAW must contain enough information for the NIU to route the data to the framebuffer on the remote ring and for the framebuffer to place the data in its memory. As shown in Fig. 1(b), 4 bits of the RAW are unused by the framebuffer, and therefore can be used by the sending NIU to route the message.

When a ring message with peripheral style addressing arrives at an NIU, bits 21-24 (4 bits) of the RAW will be used as an index in a table. Each entry of this table will contain the following three items:

- a new 7-bit ring node address
- a 13-bit protocol-specific value (e.g. port number)
- a 32-bit HIPPI I-field global network address

The new 7-bit ring node address from the table will be substituted into the RAW replacing the ring node address used to reach the NIU. The modified RAW and all data words in the message will be encapsulated in a TBDP transfer unit. A 13-bit destination port number, connection number, or other TBDP-specific information will be taken from the table. The entire TBDP transfer unit will be sent to the network node identified by the HIPPI I-field from the table.

In the case of messages destined for another PXPL5 ring, the receiving NIU will extract the message from the TBDP transfer unit. The resulting message will be placed on the ring without any further processing. The messages bound for the Y, MP, or MWS do not need a 7-bit ring node address. In such cases, these 7 bits could be ignored by the receiver or may be used to convey other control information. If the ring node address field is ignored, the substitution from the table could be omitted. This look-up table will be referred to as the **Network Address Table (NAT)**, and the index in the NAT will be known as the **Peripheral Address Table Index (P-ATI)**.

3.3.2.2 Host style network-bound communication.

Host style communication is used where more addressability is desired than is available with the 4-bit table index of the peripheral style. This style is based on the message format used by ROS as shown in Fig. 1(d). The 12-bit "mailbox" field of the address word will be used to index a table. Each entry in the table will contain the following four items:

- a new 7-bit ring node address

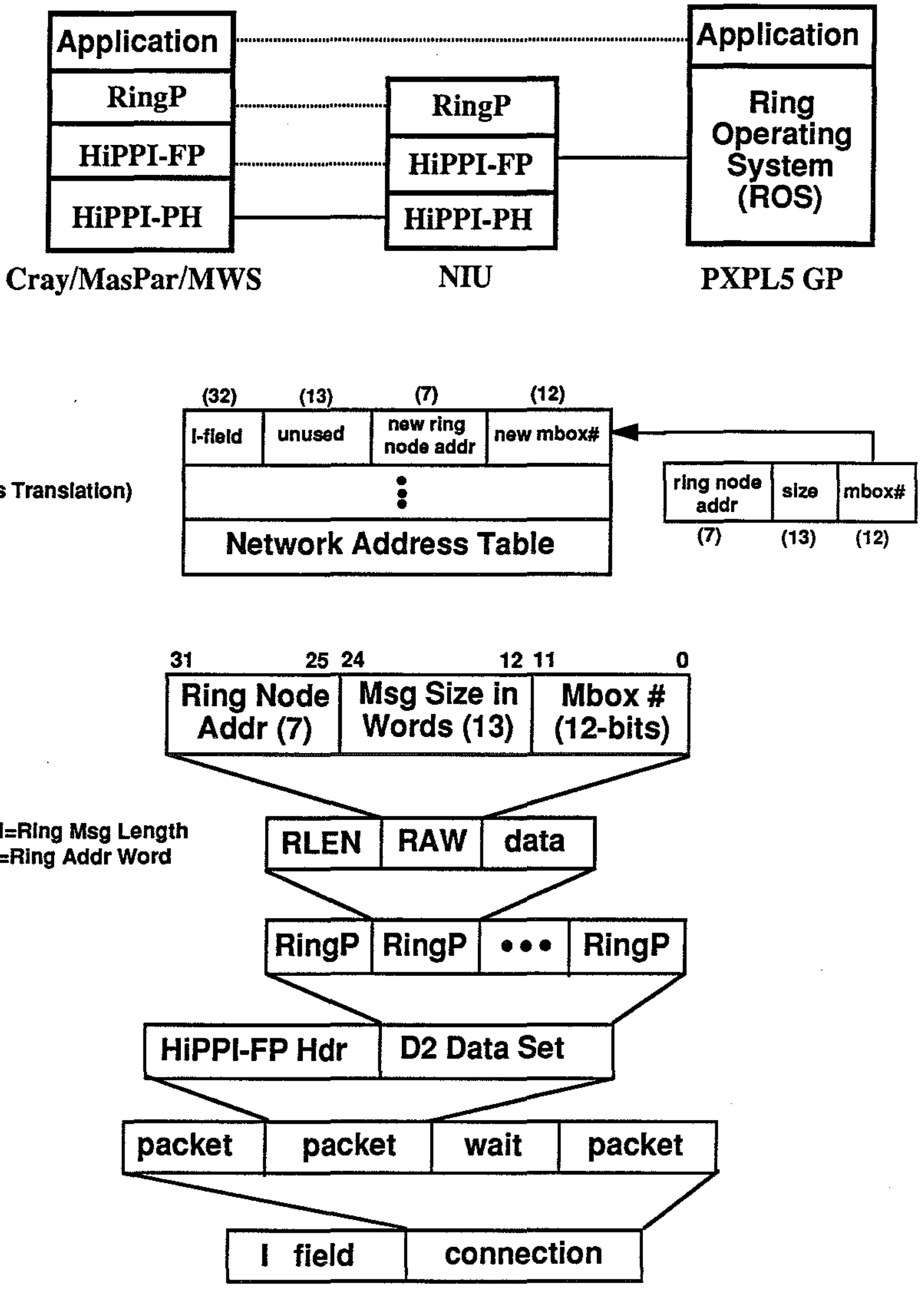

Fig.6: Host Style Communication and Associated Packet Format

- a new 12-bit mailbox number
- a 13-bit protocol-specific value
- a 32-bit HIPPI I-field value

The new 7-bit ring node address from the table will be substituted into the RAW replacing the ring node address used to reach the NIU. The new mailbox number will be substituted into the RAW replacing the table index. The new mailbox number specifies a mailbox located on the remote host. The modified RAW and all data words in the message will be encapsulated in a TBDP transfer unit. The destination port number of the TBDP segment will be taken from the table. The entire TBDP transfer unit will be sent to the Network node identified by the HIPPI I-field from the table.

In the case of messages destined for another PXPL5 ring, the receiving NIU will extract the message from the TBDP transfer unit. The resulting message will be placed on the ring without any further processing. Messages bound for the Y, MP, or MWS need not have the 7-bit ring node address replaced. This substitution can either be omitted, or used to convey other information.

This table will be known as the **Network Address Table (NAT)** and its index as the **Host Address Table Index (H-ATI)**. The Host and Peripheral modes will use the same NAT with 4096 entries.

3.3.3 Command port messages

In addition to the two styles of network-bound messages, there are messages that terminate at the NIU itself. The latter type of messages support functions like **testing, initialization**, and **monitoring**. We will refer to this type of messages as **control messages**. The ring port 0 will be exclusively used for control messages.

Messages sent to the control port will be decoded by software running on the microprocessor. The RAW of each control message will include an opcode indicating the type of the control message. Opcodes will be assigned as needed by particular NIU programs. Opcodes to be assigned will include functions for loading programs into the NIU, examining NIU memory, querying status, and setting software parameters.

The EPROM bootstrap code for the microprocessor will implement the required opcodes for downloading the network software and some minimal system testing.

Standard protocol software will need to assign control message opcodes to select between the two addressing styles discussed above. When a ring message arrives at the NIU interface port, we must a priori know the RAW format of the message. The existing RAW formats that led to the two addressing styles, described above, leave us no room to indicate the type in the message itself. The host and peripheral style data messages will be intermixed with control messages indicating the style of the data messages to be transmitted next. It will be the responsibility of ROS and application software to send these control messages when necessary. If a host style message arrives at the data port when a peripheral style message is expected or vice-versa, it may result in transmission of the message to an incorrect destination.

3.4 Protocol Processing

The protocol handling subsystem further consists of the following subsystems:

- Ring bound data FIFOs
- Network bound data buffers
- High-performance microprocessor
- Processor RAM/ROM
- Checksum computation units

The Cypress CY7C611 SPARC integer unit has been selected as the NIU's microprocessor for its low context switch overhead and simpler hardware design.

3.4.1 Data Buffers

3.4.1.1 Ring bound FIFO buffers: The incoming data from the HIPPI subsystem will be placed in a FIFO buffer. There will be two FIFO buffers for efficiency. The buffer accesses will be interleaved, i.e., while one buffer is being filled by the HIPPI subsystem, the ring interface will empty the other. The following more detailed discussion assumes that a HIPPI connection has already been established at the destination port.

When a packet arrives at the HIPPI destination port, a data buffer controller will direct it to one of the two buffers. As soon as the first word is received, the buffer controller will indicate to the microprocessor that a header is arriving . The microprocessor will proceed to read and process the header while the rest of the packet is being written into the FIFO. The software can read a variable number of words from the FIFO making up the packet header. Any data not read by the processor can later be transferred to the ring.

The ULP-id byte of the HIPPI-FP header will be used to determine the ultimate destination of the data. Multiple ULP-ids will be assigned to NIU to support services such as loop-back test and data transfer to the ring via multiple upper level protocols.

If examination of the header by ULP software reveals that a potentially valid packet is arriving, it will wait until the entire packet has been received. At this point, an error flag indicating parity and/or LLRC error is available from the HIPPI subsystem. The checksum unit will provide the computed checksum for the packet received. If these two checks are successful, the microprocessor immediately flags the ring interface subsystem to start transferring the data from buffer to the ring. If any of the data validity tests (parity, LLRC, header or checksum) on the packet fails, the processor will flag the buffer controller to discard the packet and not place it on the ring.

To clarify the above discussions, each of the two FIFO buffers can be in one of the following states: (1) receiving data from the HIPPI subsystem only; (2) receiving HIPPI data with the processor reading the header; (3) waiting for the processor to determine if a packet was good; (4) sending data to the ring; (5) waiting to receive the HIPPI data. The buffer controller will maintain these states for the two buffers in cooperation with the protocol software.

3.4.1.2 Network-bound SRAM buffers: Data from the ring will be placed into a buffer system similar to that used on the ring bound side. As the ring message comes in, a partial checksum will be computed and its length determined. The RAW of the message will be captured into a separate register and not placed in the buffer. The protocol processor will be flagged that RAW is available.

While the remainder of the message from the ring is being written into the buffer, the protocol processor begins constructing headers for the HIPPI-FP and ULP layers. Table look-up in the NAT can be performed during this time. When the complete message is stored in the buffer, the message length and partial checksum is made available to the protocol processor. Header formation is completed at this time. The header is then written into the header FIFO and the data buffer controller is signaled to begin sending the packet to the HIPPI subsystem.

Multiple buffers will be implemented to support research into high performance reliable transport protocols. A reliable protocol will use a positive acknowledge and retransmit scheme with sliding windows, requiring buffers for all unacknowledged packets. The microprocessor will have control over buffer allocation, and therefore software can reuse buffers when acknowledgements are received. When the protocol software needs to do a retransmission, it must re-write the header into the header FIFO but the data part of the packet remains stored in the same buffer. The number of buffers is based on the calculation of section 2.3.1 using buffering capacity of the network. A timer facility will be provided for protocol software to implement time-outs for packet retransmission.

3.4.2 Checksum computation hardware and software

In order to keep up with the high data rates, the packet checksum computation must be performed in hardware. We intend to support a 16-bit checksum, as used in TCP. This type of checksum, however, is not most suited for hardware implementation in NIU's environment with 32-bit data paths. Furthermore, protocols implemented during later research may handle checksums on their headers differently.

The checksum computation hardware will add up two independent 16-bit checksums over each half of the 32-bit word in the data stream. The entire HIPPI packet will be added on ring bound data, and the entire ring message on network-bound data.

The hardware checksum units will provide the two 16-bit partial sums to the protocol processor. Software on the processor will make protocol-specific adjustments to the checksum. Such adjustments may include subtracting the partial checksum over the header section from the checksum received from the hardware unit to arrive at the final checksum value. Based on the checksum, the protocol software will determine if the ring-bound data can be passed on to the ring interface. On network-bound data, the protocol software will insert the adjusted checksum into the header.

Protocols that do not use a checksum can ignore the header computation units. Checksum schemes that can not be adapted to use these 16-bit hardware checksum adders are incompatible with the NIU. Protocols that require incompatible checksum schemes can not be implemented on the NIU without modification to the protocol.

3.4.3 NIU Testability

Additional data paths will be provided between the microprocessor and data buffer memories so that these components can be tested in isolation. We will develop diagnostic software for the NIU to verify proper operation of these components. These data paths will be available to protocol software, which could use them to implement performance test features. Desirable test features of protocols include loop back tests, data sinks, and data sources.

4. NIU AND FUTURE RESEARCH

The hardware discussed in the preceding sections provides an adaptive platform for communications related research. The software running on the microprocessor will control the data buffering hardware to implement several network protocols.

The first protocol software written will be essentially "no protocol." The RingP delivery service layer will run directly above the HIPPI-FP layer. Future protocol research will be carried out by writing new software running in the NIU. A possibility includes implementation of the full HIPPI standard stack of HIPPI-LE, IEEE-802.2, IP, and TCP.

ACKNOWLEDGMENT

The authors would like to thank Tom Sandoski, Tim Seever, Dan Stevenson, Dan Winkelstein, and Jon Whaley of MCNC for technical discussions and Sharon Walters for help with the manuscript.

REFERENCES

1. H. Fuchs et. al., "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories", *Computer Graphics*, Vol. 23, No. 3, 1989, pp. 79-88.

2. ANSI Standard for Information System, "HIPPI Mechanical, Electrical, and Signalling Protocol Specification (HIPPI-PH)", X3T9.3/88-023 Rev 7.2, August 2, 1990.

3. ANSI Standard for Information System, "HIPPI Framing Protocol Specification (HIPPI-FP)", X3T9.3/89-013 Rev 4.2, June 24, 1991.

4. SPARC RISC User's Guide, Cypress Semiconductor, 2nd Edition, February, 1990.

5. APPLICATIONS HANDBOOK, Cypress Semiconductor, August, 1990.

APPENDICES

The following sections outline the details of software architecture and hardware register definitions. These details will be updated as the system design and implementation is further developed.

A. NIU SOFTWARE ARCHITECTURE

NIU's software architecture will be object oriented where each object maps onto a corresponding hardware module. A real-time supervisor kernel will provide prioritized scheduling of service requests from various subsystems on the board. The service request can be generated either by setting a bit in a status register or via a hardware interrupt.

The modular software architecture will facilitate network research by allowing the protocol stacks to be changed with minimal impact on the application. A raw or no-protocol stack will be implemented first. The hardware and software architecture has been designed to support TCP/IP as the target protocol and will be implemented next. The Fig. A1 shows the software hierarchy.

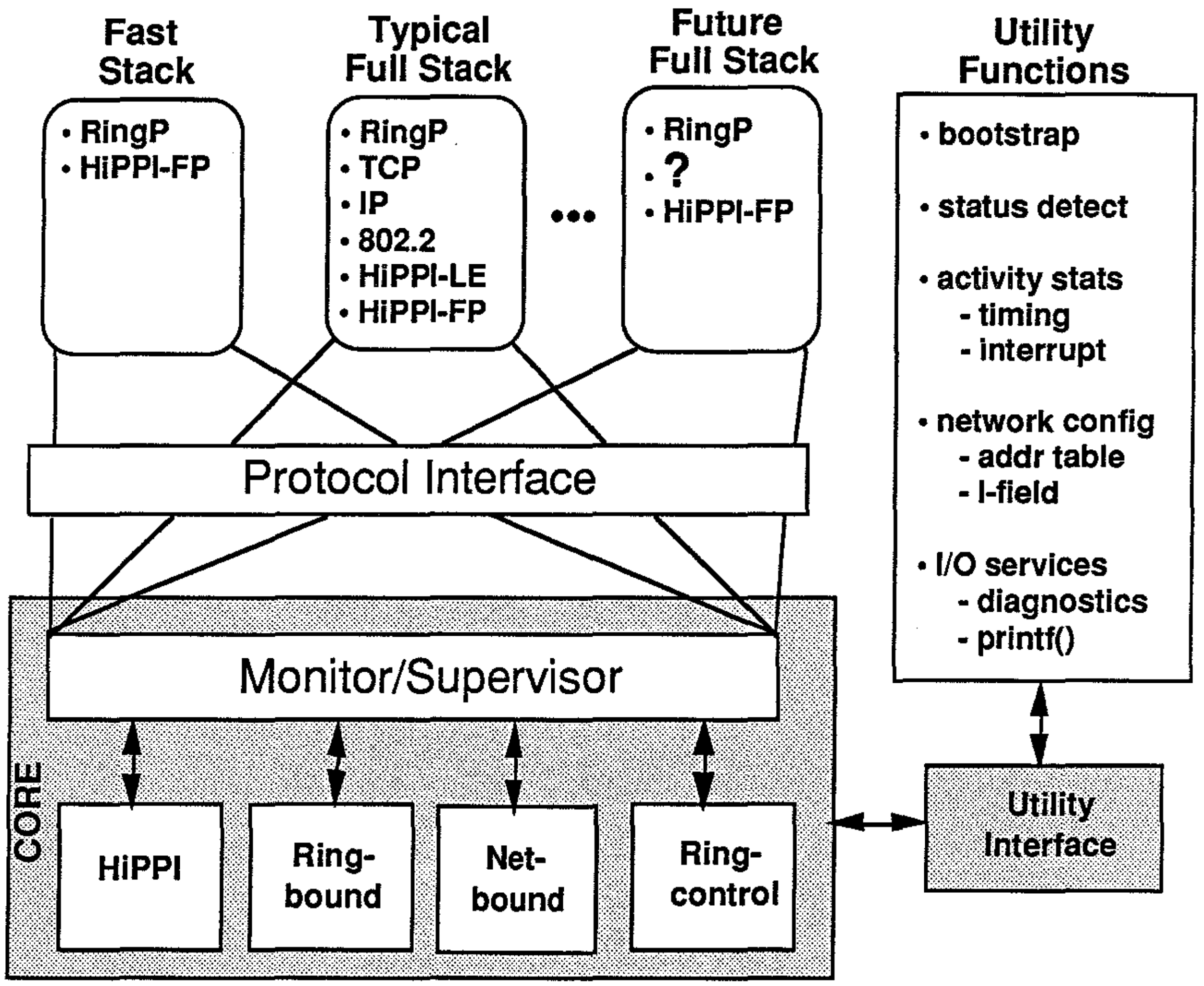

Fig. A1: NIU Software Architecture

B. NIU HARDWARE REGISTERS

1.0 Interrupts

Interrupt Level	Priority	Polling Equivalent	Activated By
RINGBOUND	0x7	PKTARRIVING*x* to 1	RBINTENABLE
RINGBOUND	0x7	PKTARRIVED*x* to 1	RBINTENABLE
NETBOUND	0x6	DPRECEIVING of RINGCTRL to 1	DPARRVINTENABLE
NETBOUND	0x6	DPRECEIVING back to 0	DPININTENABLE
DATAPORTWRITE	0x5	DATAPORTBUSY to 1	DPTXINTENABLE
HIPPISRCWRITE	0x4	SENDINGPKT of HIPPICTRL to 0	HSINTENABLE
COMANDPORT	0x3	No equivalent to msg arrived interrupt	CPRXINTENABLE
COMANDPORT	0x3	CMDPORTREADY of RINGCTRL to 1	CPTXINTENABLE
CLOCKOVERFLOW	0x2	High bit of CLOCK in NIUSTATUS to 0	Always activated.

2.0 Registers

Some of the registers in the following table are labelled "trigger". This implies that they are activated by performing a write to them. The value written is ignored, however, write enable triggers the action controled by that register.

Register	Address	Trigger	Hardware Signal
CLEARERROR	0xffffff54	Yes	PPClrErr
CMDPORTCLOSE	0xffffff40	Yes	CPEndPktEnL
CMDPORTFIFO	0xffffff04		CPFifoRdEnL
CMDPORTWRITE	0xffffff04		CPCmdWrEnL
DATAPORTABORT	0xffffff58		DPResetEnL
DATAPORTLEN	0xffffff24		DPLenRdEnL
DATAPORTOPEN	0xffffff5c	Yes	PRcvEnL
DATAPORTRAW	0xffffff20		DPRawRdEnL
DATAPORTXMIT	0xffffff4c		DPTxGoEnL
HIPPICTRL	0xffffff0c		HPCsr{Rd,Wr}EnL
HIPPIIFIELD	0xffffff10		HSIFWrEnL,HDIFRdEnL
HIPPILEN	0xffffff24		HSLenWrEnL
HIPPIXMIT	0xffffff50	Yes	HSTxGoEnL
NETBNDCHKSUM	0xffffff28		NBCksmRdEnL
NETBNDCTRL	0xffffff2c		NBCsr{Rd,Wr}EnL
NETBNDHEADER	0xffffff18		NBHeadWrEnL
NIUSTATUS	0xffffff00		PPStatRdEnL
RESETFIFO0	0xffffff44	Yes	RB0RstEnL
RESETFIFO1	0xffffff48	Yes	RB1RstEnL
RINGBNDCHKSUM	0xffffff14		RBCksmRdEnL
RINGBNDCTRL	0xffffff30		RBCsr{Rd,Wr}EnL
RINGBNDFIFO0	0xffffff18		RB0HeadRdEnL
RINGBNDFIFO1	0xffffff1c		RB1HeadRdEnL
RINGCTRL	0xffffff08		RPCSRRdEnL
RINGP	0xffffff34		P1DRdEnL

CLEARERROR	–	Trigger this register to clear the ERRORRESET bit in NIUSTATUS.
CMDPORTCLOSE	–	This trigger closes the channel acquired by the Command Port allowing another channel to be acquired.

CMDPORTFIFO	–	Reads from this register return the next word in the Command Port FIFO and remove that word from the FIFO. Valid only when CMDFIFOAVAIL is set.
CMDPORTWRITE	–	The first word written to this register is used as Ring Address Word (RAW) to acquire a ring channel. Successive words are transmitted on this channel until CMDPORTCLOSE is triggered.
DATAPORTABORT	–	Trigger this register to stop the RingP hardware from attempting to acquire a ring channel. The next FIFO read is assumed to be the beginning of a new RingP packet.
DATAPORTLEN	–	Length of the last packet to enter the data port. Valid from the second NETBOUND interrupt until next trigger of DATAPORTOPEN. Only the lowest 16 bits are used. Use DATAPORTLEN_MASK to mask out invalid bits.
DATAPORTOPEN	–	Trigger the Data Port to accept the next ring message sent to it over the PXPL5 ring.
DATAPORTRAW	–	The first word of the packet entering the data port. Valid from first NETBOUND interrupt until next trigger of DATAPORTOPEN.
DATAPORTXMIT	–	Trigger this register to start the RingP hardware reading the buffer indicated by the RBFIFOSELECT bit.
HIPPICTRL	–	The control and status register for the HIPPI interface
HIPPIIFIELD	–	Write I-Field of a remote destination to this register before raising MAKEREQUEST bit in HIPPICTRL.
HIPPILEN	–	Write the length minus 1 of the data in the window the Source Port is reading from. A zero causes one word to be read from the window.
HIPPIXMIT	–	Trigger the Source Port to send a packet. Must be connected, i.e. have ACCEPTED raised.
NETBNDCHKSUM	–	Adds both halves of the incoming data. Does not include RAW. The low 2-bytes contain the sum of all the low 2-bytes in the incoming data. The high 2-bytes sum the high 2-bytes.
NETBNDCTRL	–	The control and status register for the netbound buffers
NETBNDHEADER	–	Words written to this address are stored in the Netbound FIFO until the Source Port reads them. Should not be written to when the FIFO is full or transmitting to the source.
NIUSTATUS	–	The control and status register for the Processor Subsystem
RESETFIFOx	–	These triggers clear their respective Ring Bound FIFOs to allow another HIPPI packet to enter.

RINGBNDCHKSUM	–	Ring Bound Check sum can be read after PKTARRIVED comes on. The left and right shorts of the packet are summed into the right and left halves of this register. The lower half does not overflow into the upper half. If CHKSUMSTATE is SUMERROR, too many packets have come in and a check sum has been lost. If CHKSUMSTATE is SUMQUED0, this register contains an invalid value.
RINGBNDCTRL	–	The control and status register for the Ring Bound FIFOs
RINGBNDFIFO*x*	–	Reads from the register return the last value in Ring Bound FIFO x and remove that word from the FIFO. Only valid if FIFO x is selected to be read from the processor and is not empty.
RESETFIFO*x*	–	These triggers clear their respective Ring Bound FIFOs to allow another HIPPI packet to enter.
RINGP	–	The register holds the last value the RingP hardware read from the Ring Bound FIFOs unless the RINGPCOUNTER bit is raised in which case is holds the RingP counter value.

3.0 Register Bit Fields

3.1 HIPPICTRL Bit Fields

The control and status register for the HIPPI interface.

HIPPICTRL Field	Bit Mask	Low Active	Read/Write	Hardware Signal
ACCEPTED	0x00020000		Read	HSConnectH
CONNECTREQUEST	0x00000001		Read	HDRequest
DSTINTERCONNECT	0x00000004		Read	RXINTCSDH
HAVEPULSES	0x00100000		Read	SRdyPosH
HIPPICONNECT	0x00000100		Write	RXCONNECT
HSINTENABLE	0x00020000		Write	HSIntEnab
IFLDPARITY	0x00000078		Read	HDPErrL[0..3]
MAKEREQUEST	0x00010000		R/W	HSRequestH
PULSECOUNT	0x0000003f		Write	
PULSEZERO	0x00000002		Read	RxCzero
REJECTED	0x00040000		Read	HS RejectH
SENDINGPKT	0x00200000		Read	HSPacket
SOFTLED0	0x01000000	Yes	Write	
SOFTLED1	0x02000000	Yes	Write	
SOFTLED2	0x04000000	Yes	Write	
SOFTLED3	0x08000000	Yes	Write	
SRCINTERCONNECT	0x00080000		Read	TXINTCDSH

ACCEPTED	-	Bit is 1 while the remote destination accepts the connection. Valid after MAKEREQUEST is raised.
CONNECTREQUEST	-	Bit is 1 while the remote HIPPI Source requests a connection or wants to continue a connection.
DSTINTERCONNECT	-	Bit is 1 while there is a physical connection to a remote Source.

HAVEPULSES	-	Bit is 1 while the Source Port has burst request pulses it can respond to.
HIPPICONNECT	-	Set bit to 1 to complete a HIPPI connection to the Destination Port.
HSINTENABLE	-	Set bit to 1 to enable HIPPISRCWRITE interrupts.
IFLDPARITY	-	All bits of this field are 1 when the parity is correct across the I-Field word. Valid while CONNECTREQUEST is raised.
MAKEREQUEST	-	Set bit to 1 to request a connection from an interconnected remote destination. Set bit to 0 to break down connection. Current value is readable.
PULSECOUNT	-	Write to this field the number of burst request pulses to be sent to the remote Source.
PULSEZERO	-	Bit is 1 whenever the Destination Port has no burst request pulses to send.
REJECTED	-	Bit is 1 when a remote destination refuses the connection request.Valid after MAKEREQUEST is raised.
SENDINGPKT	-	This bit becomes 1 when the Source Port is triggered and returns to 0 after a packet is sent.
SOFTLEDx	-	Set bit to 0 to turn on LED. LED0 is the inner light and LED3 is the outer light of the four lights. Note that writes to HIPPICTRL must maintain the other writable fields in the register.
SRCINTERCONNECT	-	Bit is 1 while the source cable is connected to a remote destination.

3.2 NETBNDCTRL Bit Fields: The control and status register for the netbound buffers

NETBNDCTRL Field	Bit Mask	Low Active	Read/Write	Hardware Signal
BANKSELECT	0x00000040		Write	BankSel
HEADERACTIVE	0x00000004	Yes	Read	HdrActiveH
HEADEREMPTY	0x00000001	Yes	Read	NBHEFL
HEADERFULL	0x00000002	Yes	Read	NBHFFL
NETBNDWINDOW0	0x00000007		Write	NBMWin0
ETBNDWINDOW1	0x00000038		Write	NBMWin1
PACKETSENT	0x00000010		Read	NBEOPL
SENDDATA	0x00000080		Write	NBMDataH

BANKSELECT	-	Set this bit to 0 for incoming data to enter a window in bank 0 and for the Source Port to read data from a window in bank 1. Set to 1 to direct incoming data into bank 1 to read data from bank 0.
HEADERACTIVE	-	Bit is 1 when the Netbound Header FIFO is being read by the Source Port.
HEADEREMPTY	-	Bit is 1 when the Netbound Header FIFO is empty.
HEADERFULL	-	Bit is 1 when the Netbound Header FIFO is full.

NETBNDWINDOWx	-	Set this field to the window number (0-7) that data will enter or leave bank x.
PACKETSENT	-	Bit is 1 when the Netbound buffer window is emptied by the Source Port.
SENDDATA	-	Set bit to 1 when Source Port should send the the Netbound Header FIFO and the currently selected buffer window.

3.3 NIUSTATUS Bit Fields: The control and status register for the Processor Subsystem.

NIUSTATUS Field	Bit Mask	Low Active	Read/Write	Hardware Signal
CLOCK	0x0000ffff		Read	
ERRORRESET	0x00010000		Read	PPWasErrH

CLOCK	-	This field is the clock tick counter. It is incremented every 40 ns. Overflows cause a CLOCKTICK interrupt to occur.
ERRORRESET	-	Bit is 1 after NIU resets processor because it halted in the ERROR state.

3.4 RINGBNDCTRL Bit Fields: The control and status register for the Ring Bound FIFOs.

RINGBNDCTRL Field	Bit Mask	Low Active	Read/Write	Hardware Signal
CHKSUMSTATE	0x00003000		Read	RBChs[0..1]
SUMERROR	0x00000000			
SUMQUED0	0x00003000			
SUMQUED1	0x00002000			
SUMQUED2	0x00001000			
FIFOEMPTY0	0x00000100	Yes	Read	RB0EFL
FIFOEMPTY1	0x00000200	Yes	Read	RB1EFL
FIFOFULL0	0x00000400	Yes	Read	RB0FFL
FIFOFULL1	0x00000800	Yes	Read	RB1FFL
LONGPKT0	0x00000004		Read	RB0LongPktH
LONGPKT1	0x00000040		Read	RB1LongPktH
PKTARRIVED0	0x00000002		Read	RB0InH
PKTARRIVED1	0x00000020		Read	RB1InH
PKTARRIVING0	0x00000001		Read	RB0ArrH
PKTARRIVING1	0x00000010		Read	RB1ArrH
RBFIFOSELECT	0x00004000		R/W	RBSelH
RBINTENABLE	0x00008000		R/W	RBIntEnab
TESTBIT0	0x00010000		R/W	TEST0
TESTBIT1	0x00020000		R/W	TEST1
XMITERR0	0x00000008		Read	Pkt0ErrH
XMITERR1	0x00000080		Read	Pkt1ErrH

CHKSUMSTATE	-	This field reports the current state of the checksum hardware.
SUMQUEDx	-	These states indicate x checksum values are available to be read through the RINGBNDCHKSUM register.

SUMERROR	-	This CHKSUMSTATE indicates hardware lost a checksum value since it already held two checksum values.
FIFOEMPTYx	-	Bit is 0 when Ring Bound FIFO x is empty.
FIFOFULLx	-	Bit is 0 when Ring Bound FIFO x is full.
LONGPKTx	-	Bit is 1 when the Destination Port has attempted to buffer a packet larger than the size of Ring Bound FIFO x.
PKTARRIVINGx	-	Bit is 1 while a HIPPI packet is entering Ring Bound FIFO x
PKTARRIVEDx	-	Bit is 1 after a packet has entered Ring Bound FIFO x and is cleared when the FIFO is cleared.
RBFIFOSELECT	-	Set this bit to 0 for the RingP hardware to read Ring Bound FIFO 0 and the processor to read FIFO 1 from the RINGBNDFIFO1 register.
RBINTENABLE	-	Set bit to 1 to enable RINGBOUND interrupts.
TESTBITx	-	Set these bits to test processor connections with hardware registers.
XMITERRx	-	Bit is 1 when the packet in Ring Bound FIFO x was received with a HIPPI parity error.

3.5 RINGCTRL Bit Fields: The control and status register for the ring interface.

RINGCTRL Field	Bit Mask	Low Active	Read/Write	Hardware Signal
CMDFIFOAVAIL	0x00000010	Yes	Read	CpAvail
CMDFIFOEMPTY	0x00000001	Yes	Read	P0FEFL
CMDFIFOFULL	0x00000002	Yes	Read	P0FFFL
CMDPORTREADY	0x00000008		Read	CPTxReadyH
CPRXINTENABLE	0x00000001		Write	CPRxIntEnab
CPTXINTENABLE	0x00000002		Write	CpTxIntEnab
DATAPORTBUSY	0x00010000	Yes	Read	DPTxBusyL
DPARRVINTENABLE	0x04000000		Write	DPArrIntEnab
DPININTENABLE	0x08000000		Write	DPInIntEnab
DPRECEIVING	0x01000000		Read	DPRxBusyH
DPTXINTENABLE	0x10000000		Write	DPTxIntEnab
DPWAITING	0x02000000	Yes	Read	Rx1ReadyL
RINGPCOUNTER	0x20000000		Write	RBMOeL

CMDFIFOAVAIL	-	Bit is 1 when the processor can read a word from the CMDPORTFIFO register.
CMDFIFOEMPTY	-	Bit is 0 while Command Port FIFO is empty.
CMDFIFOFULL	-	Bit is 0 while Command Port FIFO is full.
CMDPORTREADY	-	Bit is 1 while the Command Port has a ring channel open.

CPRXINTENABLE	-	Set bit to 1 to enable COMMANDPORT interrupts to occur when ring messages arrive.
CPTXINTENABLE	-	Set bit to 1 to enable COMMANDPORT interrupts to occur when the Command Port acquires a ring channel.
DATAPORTBUSY	-	Bit is 0 while the RingP hardware is trying acquire a channel or sending a packet over an acquired channel.
DPARRVINTENABLE	-	Set to 1 to activate the arriving NETBOUND interrupt.
DPININTENABLE	-	Set to 1 to activate the arrived NETBOUND interrupt.
RINGBNDCTRL	-	The control and status register for the Ring Bound FIFOs.
RINGCTRL	-	The control and status register for the ring interface.
DPRECEIVING	-	This bit is 1 while a ring packet is entering the Data Port
DPTXINTENABLE	-	Set bit to 1 to activate DATAPORTWRITE interrupt.
DPWAITING	-	This bit is 1 after a complete ring packet is received and 0 after DATAPORTOPEN is triggered.
RINGPCOUNTER	-	Set bit to 1 to read the RingP counter from the RINGP register. Set bit to 0 to read from the RINGP register the last word RingP read from the selected Ring Bound FIFO.

C. NIU PROM SERVICES

When Pixel Planes 5 is reset, the NIU starts executing its PROM program. This program monitors the Command Port for command messages from the ring. The command messages allow the NIU software to be downloaded and execution to switch to the new program.

RAW Bit Field	Mask
Command	0x0000000f
Length	0x000ffff0
NIU Ring Address	0xfe000000

The Ring Address Word of command messages has the command in the low 4 bits, an optional length in next 16 bits. The possible commands are:

Command Name	CMD
NIUROM_PING	0
NIUROM_READ	1
NIUROM_WRITE	2
NIUROM_JUMP	3
NIUROM_LED	4
NIUROM_STATUS	5
NIUROM_VERSION	6
NIUROM_MTEST	7

NIUROM_PING	-	This command requests the NIU to send the second word of the packet back out the command port as a packet in and of itself.

Format Word	Description
RAW	Command is NIUROM_PING. Length is ignored.
Reply RAW	1 word packet to be sent out Command Port

Reply	Description
Reply RAW	From command packet

NIUROM_READ - Reads from NIU memory.

Format Word	Description
RAW	Command is NIUROM_READ. Length is number of words to read.
Reply RAW	1 word packet to be sent out Command Port
Address	Address to read data from

Reply	Description
Reply RAW	From command packet.
Data words	Length words of data.

NIUROM_WRITE - Write data in command packet to memory. The NIU does not send a reply to this message.

Format Word	Description
RAW	Command is NIUROM_WRITE. Length is how many words to write to memory.
Address	Where to write remainder of packet
Data Words	Length words of data.

NIUROM_JUMP - Command NIU processor to start executing at a new address. The NIU does not send a reply to this message.

Format Word	Description
RAW	Command is NIUROM_JUMP. Length is ignored.
Address	Where execution should jump to.

NIUROM_LED - This command sets the NIU software controlled LEDS. The NIU does not send a reply to this message.

Format Word	Description
RAW	Command is NIUROM_LED. Length is ignored.
Display	The low four bits of this word are displayed on the LEDs. Bits set to one indicate the respective LED should be turned on.

NIUROM_STATUS - Request the NIU to send back a status message.

Format Word	Description
RAW	Command is NIUROM_STATUS. Length is ignored.
Reply RAW	RAW to which status message is sent.

Reply	Description
Reply RAW	From command packet with length set to length of reply.
Board ID	24-bit magic number identifying the NIU board and an 8-bit board serial number (actually an EPROM serial number; there is no separate Board-ID chip)
Self Test Result	00000000 - all passed. 00000001 - last reset was from cpu ERROR state. 00000002 - EPROM checksum failure 00000004 - short RAM test failure 00000008 - I/O register loopback test 00000010 - Network data handling status register sanity check
Clock Counter	The low 16 bits are the current value of the clock tick counter.
Clock Overflows	Count of counter overflows since the last reset.
Trap Base Register	Value of the SPARC %tbr at the last reset

NIUROM_VERSION - Request the NIU PROM version.

Format Word	Description
RAW	Command is NIUROM_VERSION. Length is ignored.
Reply RAW	RAW to which status message is sent.

Reply	Description
Reply RAW	From command packet with length set to length of reply.
Version String	Null terminated string containing the RCS IDs for the PROM source files.

NIUROM_MTEST - This command starts the continuous memory test. Board must be reset to halt test. Errors detected by the test are indicated on the software controlled LEDs.

Format Word	Description
RAW	Command is NIUROM_MTEST. Length is ignored.