```
  SSSSSSSS    PPPPPPPP      EEEEEEEEEE      CCCCCCCC
  SSSSSSSS    PPPPPPPP      EEEEEEEEEE      CCCCCCCC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
SS            PP      PP    EE            CC
  SSSSSS      PPPPPPPP      EEEEEEEE      CC
  SSSSSS      PPPPPPPP      EEEEEEEE      CC
        SS    PP            EE            CC
        SS    PP            EE            CC
        SS    PP            EE            CC
        SS    PP            EE            CC
SSSSSSSS      PP            EEEEEEEEEE      CCCCCCCC
SSSSSSSS      PP            EEEEEEEEEE      CCCCCCCC



LL            PPPPPPPP      TTTTTTTTTT                    11
LL            PPPPPPPP      TTTTTTTTTT                    11
LL            PP      PP        TT                      1111
LL            PP      PP        TT                      1111
LL            PP      PP        TT                        11
LL            PP      PP        TT                        11
LL            PPPPPPPP          TT                        11
LL            PPPPPPPP          TT                        11
LL            PP                TT                        11
LL            PP                TT                        11
LL            PP                TT            ....        11
LL            PP                TT            ....        11
LLLLLLLLLL    PP                TT            ....    111111
LLLLLLLLLL    PP                TT            ....    111111



*START* Job SPEC Req #694 for EGB    Date  3-Dec-82  1:53:19 Monitor: Rational M
File RM:<MICRO-ARCH.IO-ADAPTER>SPEC.LPT.1, created:  8-Oct-82  1:52:08
        printed:  3-Dec-82  1:53:19
Job parameters: Request created: 3-Dec-82  1:52:36   Page limit:144   Forms:NORMA
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A
```

Functional Specification of the I/O Adaptor

DRAFT 2

1. Summary

The R1000 I/O adaptor board (IOA) connects an I/O processor (IOP) to the R1000's interprocessor bus, and supports that IOP in managing certain aspects of R1000 cluster operation. This document specifies the operational, programmatic, and physical interfaces to the IOA's two constituent subsystems. Implementation of these subsystems is discussed at the block-diagram level, using references to the R1000 block diagram (daisy file /R1000) and the I/O adaptor block diagram (daisy file /R1000/IOADAPTER0). Readers are presumed to be familiar with the RMI program architecture and the R1000 machine organization. Full understanding of the IOA's interface to its IOP requires some familiarity with the DEC Unibus and PDP-11 architecture.

2. Introduction

An R1000 cluster consists of one to four processors (named P0 through P3) and one to four I/O subsystems (named IOS0 through IOS3); a minimum cluster contains P0 and IOS0. Processors and I/O subsystems, collectively referred to as nodes, communicate at 40 megabytes per second via an 8-byte parallel path known as the Sysbus. Sysbus communication is regulated by a three-layer protocol, utilizing a small set of bussed control signals.

A cluster's I/O subsystems contain all of its peripherals, which typically include magnetic disk and tape drives, data communication devices, terminals, and printers. The intelligence to manage these peripherals is provided by an I/O processor (IOP) located in each I/O subsystem. The I/O adaptor in each subsytem provides the physical path between the Sysbus and the subsystem's IOP.

IOS0 has the additional responsibility of controlling and monitoring the cluster's operator panel, multiplexing operator communications between a local console and the remote diagnostic center, timekeeping, generating system clocks, and managing cluster startup, shutdown, auto-verification, and auto-fault isolation. The I/O adaptor in this subsystem (IOA0) provides direct support for these functions.

Physically, the IOA is composed of four subsystems. The Interprocessor Communication (IPC) subsystem provides the control and buffering necessary to interface with the Sysbus in conformance with its protocol. The Cluster Management (CM) subsystem includes physical interfaces to several cluster resources; this subsystem is required in full only by IOA0, but is benign if present on IOA1, IOA2, or IOA3. The IOP Interface provides the basic Unibus interface, which enables the IOP to communicate with IPC and CM subsytems. It also electrically terminates the Unibus, and is capable of "bootstrapping" the IOP during cluster initialization. The Diagnostic Control (DC) subsystem includes the IOA's interface to the cluster diagnostic bus, its support for console multiplexing, and its self-test logic.

3. IOP Interface

The R1000 employs DEC PDP11/24 minicomputers for its IOPs. Though all processors in the PDP11 product line are compatible (more or less) with respect to program architecture, several I/O busses are now supported. The original members of the PDP11 family introduced the Unibus, a 16-bit wide non-multiplexed asynchronous data path which links the processor with both memories and I/O devices at bandwidths up to 2.0 million bytes per second. Low-end members of the family (from the PDP11/23 downward) use the Q-bus, a Unibus variant which multiplexs address and data information onto the same bus conductors. All high-end PDP11s - including VAXs - support the Unibus, but several also utilize the higher-bandwidth Massbus for magnetic disk and tape interfaces.

Given current rates of improvement in minicomputer and microcomputer technology, the IOP interface must not be tied to a specific IOP implementation. It must therefore be designed with strict adherence to Unibus interface rules, taking no advantage of electrical or timing characteristics of any specific PDP11 family member.

3.1. Unibus protocol

The PDP11 architecture utilizes the Unibus both to access memory and to manage peripheral devices. The upper 8K bytes of physical address space (referred to as I/O space) are allocated to "memory mapped" peripheral interfaces.

A typical Unibus peripheral interface contains both device control registers and device status registers which are assigned unique addresses in the I/O space. Any instruction which references memory can be used to reference these device registers. From the PDP11 processor's perspective, device status registers are usually read-only (they are updated by the interface), whereas device control registers may be either write-only or read-write.

The Unibus supports device-initiated program interrupts, giving the peripheral interface generating the interrupt the responsibility for supplying the processor with the starting address of the interrupt service routine. The address, an *interrupt vector* in DEC parlance, can be static or dynamic; an interface can, for example, select one of several vectors as a function of some state variable.

Unibus read and write transactions utilize a master/slave protocol. A processor or peripheral interface desiring to initiate a transaction first becomes *bus master* by obtaining exclusive control of the bus from a multiple-priority-level arbitor. The bus master uses the address section of the Unibus to select the transaction's *slave*, which is usually a memory or peripheral interface. The data section of the Unibus is then used to transfer data between the master and slave. The master can continue transferring data with the slave, or select a

new slave and transfer data, until some higher-priority device steals bus mastership. The Unibus provides a parity error signal, allowing the master or slave to force a processor trap if a data parity error is detected.

The PDP11 architecture and Unibus support two priority classes. The NPR class allows a peripheral interface to steal bus mastership from the processor with minimum latency - it provides what is commonly referred to as direct memory access. Priority within the NPR class is a function of the controller's physical location on the Unibus relative to the processor; closer is better. The BG class is aimed at providing a multilevel interrupt system: bus mastership is given to an interrupting controller at macro-instruction boundaries only if its priority level exceeds the priority level of the program being executed, as specified by the processor's program status word (PSW). Four BG priority levels are provided.

The Unibus physical-level protocol is implemented by a set of control signals operating in parallel with the address and data sections. The control regime is classically asynchronous. This is convenient; with suitable attention to synchronization issues, interface timing can be driven by the device rather than the processor. However, sensitivity of the protocol to both positive and negative edges of multiple control signals requires stringent electrical hygiene.

The Unibus is electrically terminated on its near-end by its PDP11 processor; the opposite end requires identical termination. Far-end termination is normally provided by installing DEC's M9312 bootstrap/terminator card in the last active Unibus slot. If the Unibus is extended to an interface not residing in a Unibus-compatible chassis, then that external interface must provide the far-end termination (or else the Unibus must be cabled from that external interface to another Unibus-compatible chassis containing an M9312).

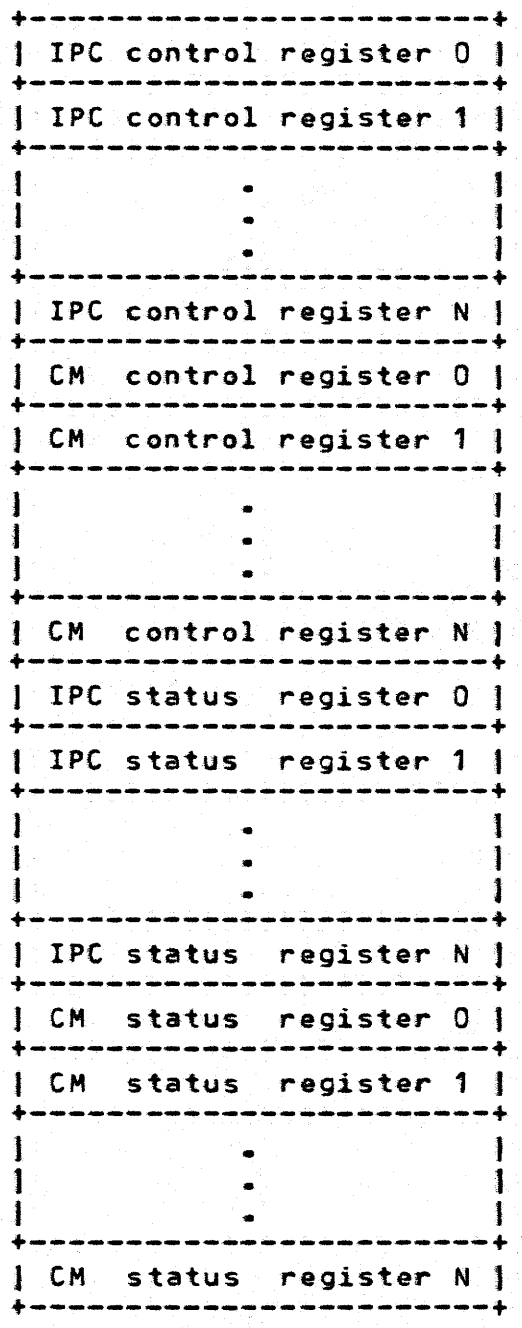
Since the termination function of the M9312 card cannot be disabled, an external interface which provides far-end Unibus termination must also provide the bootstrap function. On power-up, the PDP-11 processor "jumps" to location 24 to begin execution. The bootstrap function overides this address, forcing the processor to jump to a location in I/O space. This location maps to a ROM provided by the bootstrap function; it contains assorted self-test, console interface, and peripheral bootstrap routines.

3.2. Unibus Interface

IOA control and status registers are grouped in a block mapped to a region of IOP I/O space, as shown in Table 3-1. Throughout this document, the definition of each control and status bit will include its name, function, side effects, and location within the register block. All control bits are readable as well as writeable by the IOP, but status bits are only readable; attempts by the IOP to write status

bits will have no effect on IOP operation. The IOA modifies control bits only under direct IOP command. It does not generate or check parity for control or status registers.

Table 3-1: IOA control and status register block



Reading a status bit or changing the state of a control bit may cause

side-effects, which are defined with the specification of the bit. Since reading a control bit never initiates a side-effect, the IOP can safely apply read-modify-write instructions to control bits.

Under certain specified conditions, the IOA will generate an IOP interrupt. Part of each such condition is a control bit which enables the interrupt; at power-up, the IOA resets all interrupt enable control bits. The IOA provides a single vector for each interrupt. Each vector is treated as an array of 16 control bits, and must therefore be initialized by the IOP before the associated interrupt is enabled. Additionally, a 2 control bits representing interrupt (BG) priority level are associated with each interrupt. IOP software must never modify an interrupt's vector or priority-level unless the interrupt is disabled. Every IOA-generated interrupt (named FUBAR) has an interrupt enable control bit (named FUBAR_INTEN), an interrupt vector (named FUBAR_VECT), and an interrupt priority level (named FUBAR_PRI).

The IOA handles the packet buffers used in the IPC subsystem as if they were 64 Kbytes of standard Unibus parity memory located in Unibus (octal) addresses 400000 through 577777. The time-multiplexing scheme which allows the Unibus interface and IPC subsystem to access these buffers without conflict is described in the Sysbus Interface section of this document, together with the location of specific data packet buffers and status packet buffers.

The IOA employs a synchronous microprogrammed state machine named the Unibus microengine to implement all transaction-level and physical-level aspects of the Unibus interface. The Unibus microengine cycles once per Sysbus cycle. Its microcode is not readable or writeable by the IOP.

3.3. Termination and Bootstrapping

The IOA provides Unibus termination and IOP bootstrapping functions identical to those provided by the DEC M9312 card, as specified by its technical manual, DEC document EK-M9312-TM-002. The bootstrap ROM will occupy standard I/O space locations, and contain a power-up self-test, soft-console program, and bootstrap programs which support initial program load from magnetic disk, magnetic tape, the Sysbus, or the cluster diagnostic bus. Use of EPROMs to implement the bootstrap ROM is presently under consideration. IOP bootstrap can be initiated either by the IOA's diagnostic kernel controller, or by its standard diagnostic island.

4. Interprocessor Communication subsystem

The IPC subsystem provides the data paths and control necessary to implement the Sysbus protocol.

4.1. Sysbus protocol

The Sysbus protocol embodies three distinct layers: an upper message layer, an intermediate packet layer, and a lower physical layer; a peer-to-peer protocol is defined for each of these layers. The message layer permits objects of arbitrary length and content to be transferred among a cluster's nodes. This layer is completely implemented by software in each IOP and by software and/or microcode in each processor; no direct IOA support is provided.

Implementations of the message layer rely on the intermediate layer for *certified* transport of bounded-length *data packets* from one node to another. Certified transport means that a sender is guaranteed to receive either a positive or negative acknowledgement within a specified time interval. Certain node operations - notably page fault service - bypass the message layer and interface directly with the packet layer, thereby avoiding unnecessary overhead.

The physical layer protocol manages the flow of bits across the Sysbus itself, providing synchronization, bus arbitration, and error checking.

4.1.1. Packet layer protocol

Two packet types - *data packets*, and *status packets* - are defined, although only data packets are visible to the packet layer; status packets are fabricated and consumed by the physical layer. A data packet is composed of two parts: a header section of 16 bytes, and an information section of up to 1K bytes; a status packet consists of a 16-byte header only. Both packet types use a similarly formatted header:

Field	Length	Description
FROM_PROC	4 bits	source node's Sysbus address
TO_PROC	4 bits	destination node's Sysbus address
INFO_LENGTH	7 bits	length of the information section in half-words
SENDER_BUF	4 bits	used to match status packets to data packets
IS_STATUS	1 bit	distinguishes between data and status packets
ACK_CODE/ INFO_DESCRIPTOR	108 bits	available to the packet layer in data packets; contains an acknowledgement code in status packets

Sysbus addresses are defined in table 4-1.

Table 4-1: Sysbus Addresses

Sysbus Address	Node
0	IOA0
1	IOA1
2	IOA2
3	IOA3
4	P0
5	P1
6	P2
7	P3

Once a packet is properly formatted, the sending node's packet layer presents it to the physical layer for transmission, and starts a timeout. One of the following will then occur:

Positive acknowledge — The sending node's physical layer has received a status packet indicating that some locus of control in the receiving node has satisfactorily consumed the packet.

Negative acknowledge — The sending node's physical layer has received a status packet indicating that some locus of control in the receiving node detected an error in consuming the packet. A status code categorizing the failure is presented.

Physical layer abort — The sending node's physical layer has terminated its attempt to transmit the packet for one of the following reasons:

- it could not obtain access to the Sysbus after 16 attempts
- the destination node is broken, powered down, or does not exist
- the destination node refused to accept the packet because it is either offline or out of buffers
- the destination node detected a parity error during packet transmission

Packet layer timeout — No response was received from the physical layer within X milliseconds, implying that the destination node has failed (though its physical layer did receive the packet)

A node's physical layer copies the SENDER_BUF field in a received data packet's header into the SENDER_BUF field of the status packet it fabricates in response. This enables the sending node's physical layer to match incoming status packets with previously transmitted data packets.

To simplify recovery, the physical layer maintains a copy of the data packet in all cases other than a positive acknowledge; it can be

directed to retransmit this copy if appropriate. If the packet layer elects to abandon delivery of a packet, it must inform the physical layer so that resources maintaining the copy can be freed. Some form of packet sequence number may be needed if duplicate packets and spurious acknowledges can arise.

4.1.2. Physical layer protocol

The Sysbus contains the following signals:

Name	Source	Sink	Use
Sysbus.Data(0:63)	sender	all	bidding and packet transmission
Sysbus.Par (0:7)	sender	selected receiver	byte parity for Sysbus.Data during packet transmission
Sysbus.Pak	receiver	sender	indicates that the receiver succesfully captured the data sent during the last Sysbus cycle
Sysbus.Nak	sender or receiver	sender & receiver	immediately terminates packet transmission
Sysbus.Sender(0:7)	sender	receiver	indicates whether the Sysbus will be used for data transfer during the next cycle, and by which sender

A Sysbus cycle has the same 192 ns. duration as a processor cycle - the two are synchronized with nominally no phase shift (ignoring clock skew). To minimize the negative effects of bus turnaround and tristate bus fighting, no node may actively drive a Sysbus signal during the second quadrant of any Sysbus cycle (nominally 48 ns. through 96ns.). A node sourcing a Sysbus signal is expected to actively drive it without transition from the third quadrant of the current cycle through the end of the first quadrant of the next cycle. A node sinking a Sysbus signal is expected to strobe it at the end of each cycle.

To maximize Sysbus utilization, neither the sending node nor the receiving node can delay transmission of a data or status packet once

it has begun. Thus the physical layer transmits packets from dedicated buffers in the sending node and stores them in dedicated buffers in the receiving node. A node's packet layer may operate directly on these dedicated buffers prior to transmission or after reception, eliminating needless data movement.

A distributed arbitration mechanism known as bidding is employed to mediate Sysbus access. Since a node generates status packets involuntarily, the arbitration mechanism does not lower the priority of a node just completing a status packet transmission as it would a node just completing a data packet transmission. Thus, a node's ability to obtain the Sysbus for data packet transmission is not unfairly impaired if it happens to be a popular destination. To expedite the acknowledgement of already-transmitted data packets, the lowest-priority status packet request is given precedence over the highest-priority data packet request.

A rotating priority chain affords its requestors "fair" access by changing each node's priority after each grant of bus access. During a bid cycle, the highest-priority requesting node in the chain is granted sole access to the Sysbus. All nodes whose current priority is lower than that of this selected node increase their priority by one unit. The selected node sets its priority to the lowest possible value. Any nodes whose current priority was higher than that of the selected node leave their priority unchanged. Physically absent nodes will quickly drift into the highest priority positions in the chain, but have no effect since they never make requests.

Each node maintains a priority variable representing its current priority in the packet request chain. The values of these variables range in ascending order of priority from 0 to 7; the chain of variables is initialized at system startup such that no two variables contain the same value (this invariant must be maintained throughout normal operation). If a node has a buffered packet to transmit, the current value of its priority variable determines at which priority level it requests bus access.

The Sysbus.Sender signals allow each node to determine whether any node presently "owns" the next Sysbus cycle. If all 8 Sysbus.Sender signals are negated, then the next Sysbus cycle - referred to as a Sysbus bid cycle - is used to determine which node (if any) will become the new "owner" of the Sysbus.

A node having at least one data or status packet ready to transmit takes several actions during a bid cycle. It asserts the Sysbus.Sender signal associated with its node number, thereby indicating that the Sysbus cycle following this bid cycle will not be another bid cycle. The current value of the node's priority variable indicates which two Sysbus.Data signals will convey its data and status packet requests, according to the bit assignments are shown in table 4-2.

Table 4-2: Bidding Assignments

Priority	Request type	Sysbus.Data
0	Status	0
1	"	1
2	"	2
3	"	3
4	"	4
5	"	5
6	"	6
7	"	7
0	Data	8
1	"	9
2	"	10
3	"	11
4	"	12
5	"	13
6	"	14
7	"	15

A node with no data or status packets ready to transmit must still participate in the bid cycle: it negates the two Sysbus.Data bits associated with its current priority.

At the end of the bid cycle, each bidding node determines whether or not it has "won" by determining whether it submitted the highest-priority bid. Every node, bidding or non-bidding, examines Sysbus.Data(0:15) in order to properly update its priority variable. If there is at least one status bid, or if there are no bids at all, the priority variable is unchanged. If there are no status bids, but at least one data bid, then each node with a current priority less than the winner's increments its priority variable. The winner sets its priority variable to 0.

If there are no bids, then all Sysbus.Sender signals will be negated, indicating that the next cycle will be another bid cycle. Otherwise, the "losing" bidders negate their Sysbus.Sender signals and wait for the next bid cycle. The winner continues to assert its Sysbus.Sender signal during each cycle of its transmission except the last. The selected receiver is responsible for verifying that one and only one Sysbus.Sender signal is asserted throughout the transmission.

If a node is unable to win Sysbus ownership after 16 bid cycles, it gives up and informs its physical layer.

On the Sysbus cycle following a successful bid, the sender places the

first half-word of the packet header on the Sysbus, allowing every other node to inspect the header's TO_PROC field and thus determine if it is the selected receiver; any node detecting a parity error in the header word asserts Sysbus.Nak. The receiver stores the packet header along with the rest of the packet into a dedicated buffer, using the length field of the header to determine when transmission is complete.

Normally, the selected receiving node indicates its ability to accept the incoming packet by asserting Sysbus.Pak on a cycle by cycle basis. Alternatively, a selected receiving node may refuse to accept the incoming packet at some point in the transfer by asserting the Sysbus.Nak signal. During a Sysbus cycle following the receiver's assertion of Sysbus.Nak, the receiver places a code explaining its action onto Sysbus.Data; the sender captures this code and negates its Sysbus.Sender signal, thereby relinquishing Sysbus ownership. A newly_selected receiving node immediately aborts if it is offline (e.g. in diagnostic mode), or if it has no buffer in which to place the incoming packet. A receiving node asserts Sysbus.Nak in mid-packet if it detects a parity error on the Sysbus, or if it changes state to offline.

When the physical layer in the receiving node buffers the last half-word of the incoming packet without error, it informs its packet layer. Generally, some locus of control in the receiving node "consumes" the packet, and informs its physical layer that the buffer containing the packet is free to be reused. The physical layer then fabricates a status packet, which is transmitted back to the sender to confirm successful receipt - and in some cases, processing - of the data packet. The sender is constrained to allocate buffers in such a way that space for this incoming status packet is guaranteed. On receiving a status packet, the sending node's physical layer informs its packet layer, which frees the buffer containing the original data packet.

If the sender detects a parity error in the status packet, it asserts Sysbus.Nak, which causes the receiving node's physical layer to terminate that status packet, negate its Sysbus.Sender during the next Sysbus cycle, and retry status packet transmission. Up to N consecutive retries are attempted before the receiving node's physical layer gives up and informs its packet layer.

During transmission of a data or status packet, the transmitting node may decide to abort the transfer (perhaps due to a diagnostic event somewhere). It does this by asserting Sysbus.Nak during the current Sysbus cycle and placing a justification code on Sysbus.Data in the subsequent cycle. To avoid bus fighting if sender and receiver happen to abort simultaneously, receiver justification codes are defined to occupy Sysbus.Data(0:15), and transmitter justification codes are defined to occupy Sysbus.Data(48:63). When driving a justification code onto Sysbus.Data, a receiving node only enables drivers for the high-order two bytes. Similarly a transmitting node only enables drivers for the low-order two bytes.

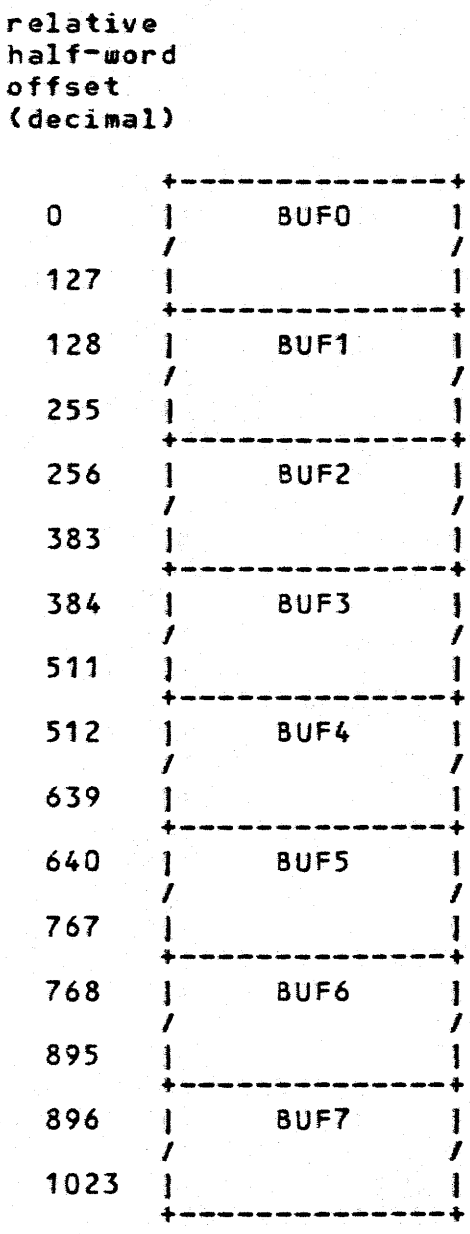
4.2. Sysbus Interface

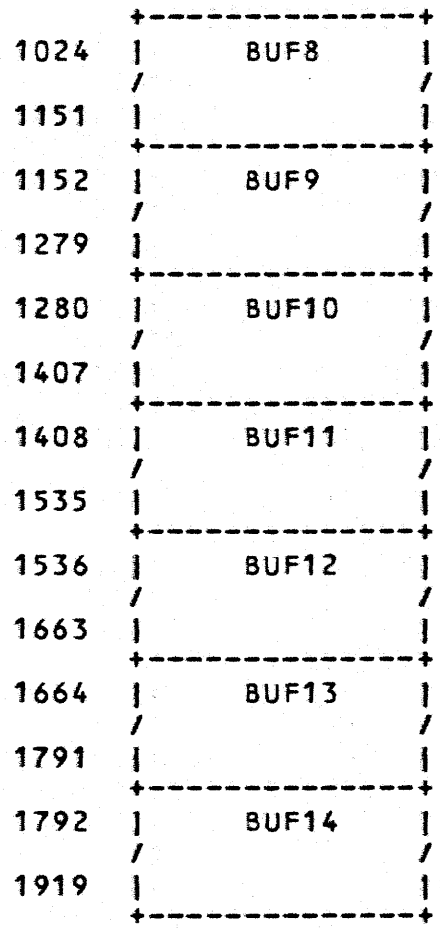
The Sysbus Interface is dominated by two elements: the packet buffers, and the microengine, named the Sysbus Controller, which manages the packet buffers and implements parts of the Sysbus packet-level and low-level protocols.

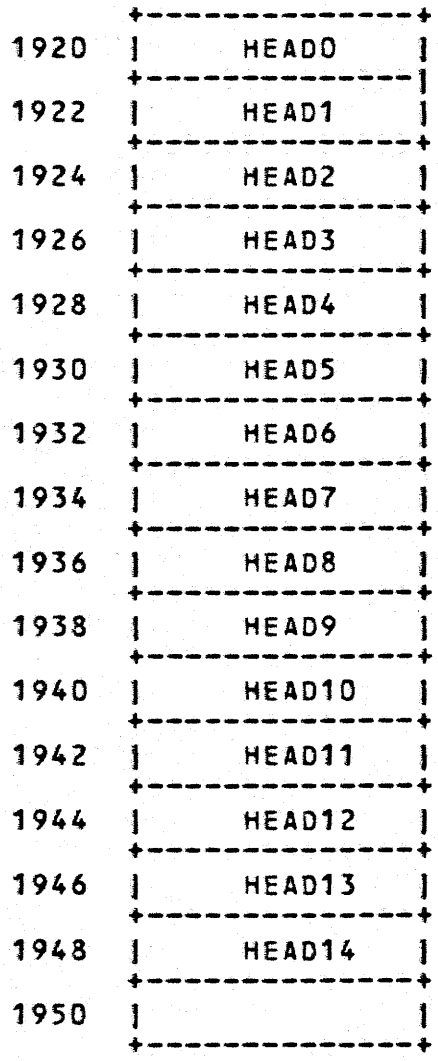
The IOA implements the packet buffers with 16K bytes of static NMOS RAM, organized as 2K 8-byte half-words; an additional array of 1K 8-bit words provides byte parity checking. This memory is logically subdivided into 16 1K-byte buffers named BUF0 through BUF15. The Unibus Interface maps each of the 64 1K-byte logical pages of IOP address space located in octal addresses 400000 through 577777 onto one of these sixteen buffers. It accomplishes this by providing each logical page with a 4-bit BUF_SEL(0..63) control field whose contents select a buffer. BUF_SEL(0..63) are located in IPC control words X through Y.

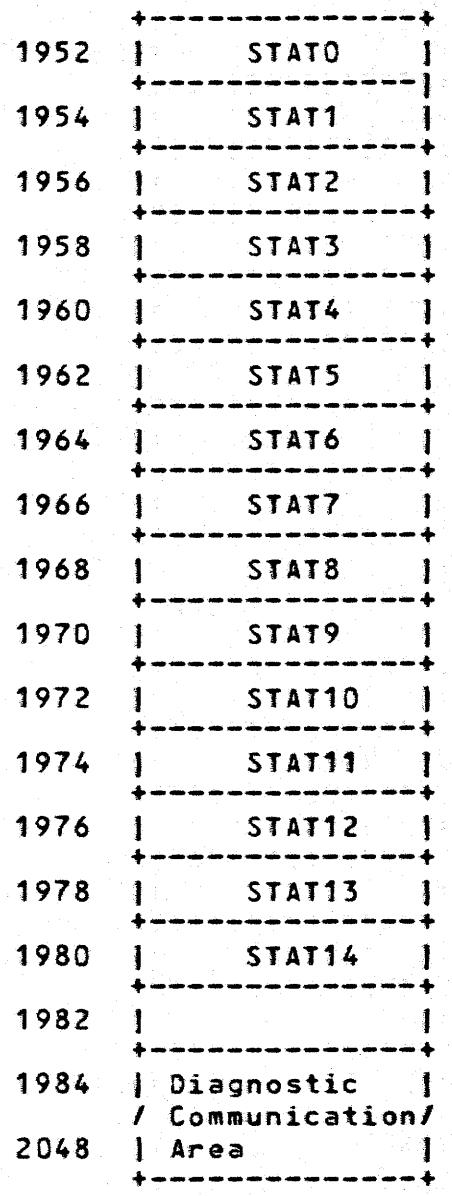
BUF_SEL(0..63) are not initialized at cluster power-up. Therefore, IOP software must establish initial values before attempting to reference any logical page associated with the packet buffers. This initialization may be included in the IOP's bootstrap program.

BUF0 through BUF14 are capable of storing a maximal-length data packet information section. The first 32 half-words of BUF15, named HEAD0 through HEAD15, buffer packet headers. The second 32 half-words of BUF15, named STAT0 through STAT15, buffer status packets. Since BUF15 is not used to store data packets, HEAD15 and STAT15 are not used. The last 64 half-words of BUF15 are used by the Diagnostic Control Subsystem. A map detailing this logical partitioning is shown below:


relative
half-word
offset
(decimal)
0
BUF0
127
128
BUF1
255
256
BUF2
383
384
BUF3
511
512
BUF4
639
640
BUF5
767
768
BUF6
895
896
BUF7
1023


1024
BUF8
1151
1152
BUF9
1279
1280
BUF10
1407
1408
BUF11
1535
1536
BUF12
1663
1664
BUF13
1791
1792
BUF14
1919


1920
HEAD0
1922
HEAD1
1924
HEAD2
1926
HEAD3
1928
HEAD4
1930
HEAD5
1932
HEAD6
1934
HEAD7
1936
HEAD8
1938
HEAD9
1940
HEAD10
1942
HEAD11
1944
HEAD12
1946
HEAD13
1948
HEAD14
1950

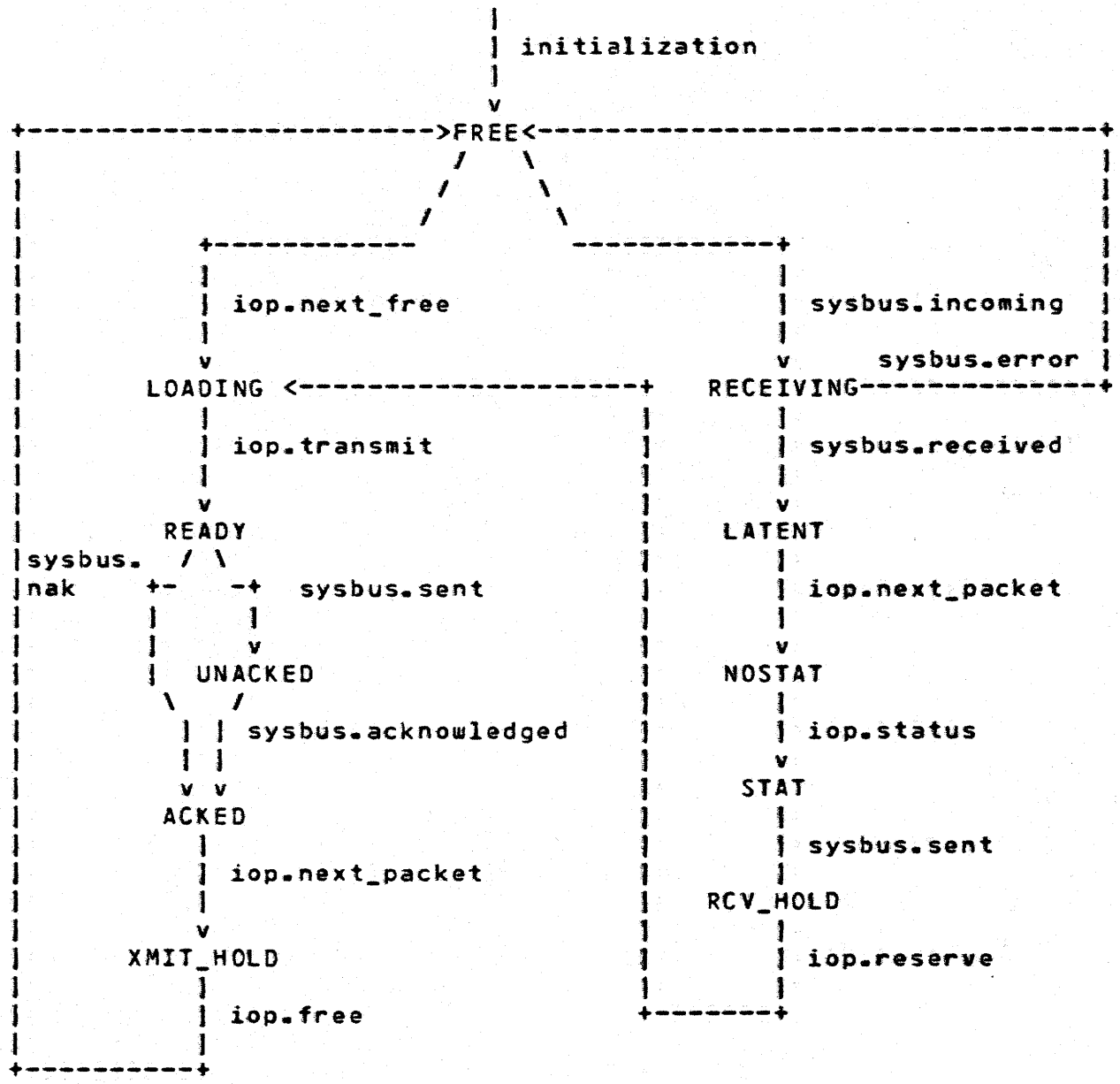

1952
STAT0
1954
STAT1
1956
STAT2
1958
STAT3
1960
STAT4
1962
STAT5
1964
STAT6
1966
STAT7
1968
STAT8
1970
STAT9
1972
STAT10
1974
STAT11
1976
STAT12
1978
STAT13
1980
STAT14
1982
1984
Diagnostic
Communication
2048
Area

The Sysbus Controller maintains a state variable for each of the 15 BUFx/HEADx/STATx buffer triplets. State names, binary codes, and definitions follow:

Name	Code	Definition
FREE	0	BUFx, HEADx, and STATx contain unneeded data, and may be allocated for transmission or reception
LOADING	1	The IOP is loading BUFx and HEADx with a data packet to be transmitted
READY	2	The data packet in BUFx and HEADx is ready to transmit
UNACKED	3	The data packet in BUFx and HEADx has been transmitted, but no status packet has yet been received
ACKED	4	The data packet in BUFx and HEADx has been transmitted; STATx contains a status packet received in response, but the IOP has not yet been informed
XMIT_HOLD	5	The IOP has been informed that STATx contains a status packet received in response to the data packet in BUFx and HEADx, but has not freed the triplet
RECEIVING	6	An incoming data packet has been partially received into BUFx and HEADx
LATENT	7	An incoming data packet has been stored in BUFx and HEADx, but the IOP has not yet been informed
NOSTAT	8	The IOP has been informed of the incoming packet in BUFx and HEADx, but has not yet fabricated a status packet in STATx
STAT	9	STATx contains a status packet to be transmitted in response to the packet in BUFx and HEADx
RCV_HOLD	10	The status packet in STATx has been transmitted

IPC status registers X through X+3 contain 15 4-bit fields indicating the current state of each triplet.

State transitions for a particular BUF/HEAD/STAT triplet are caused by IOA initialization (all state variables are set to Free), Sysbus events, and IOP commands and queries, as follows:



iop.free forces a triplet to be FREE unless it is READY, RECEIVING, or STAT

iop.reserve forces a triplet to be LOADING unless it is READY, RECEIVING, or STAT

4.2.1. Packet transmission

Transmission of a data packet begins when packet-layer software in the IOP obtains an empty buffer triplet from the IOA by reading status register X, which contains the FREE_AVAIL status bit and the 4-bit NEXT_FREE status field. If FREE_AVAIL is asserted, then NEXT_FREE identifies a free triplet; as a side effect, the state of this triplet is set to LOADING. If FREE_AVAIL is not asserted, then no free triplet presently exists. The IOA will generate the FREE interrupt anytime one or more buffers is in the FREE state. FREE_INTEN resides in IPC control register X, FREE_VECT resides in IPC control register X, and FREE_PRI resides in control register X.

When a free triplet is available, IOP software proceeds to load the BUF and HEAD portions with the data packet; it can accomplish this with memory reference instructions, or by instructing some other peripheral device to transfer the data directly with Unibus NPR transfers to the appropriate range of addresses. In either case, IOP software is responsible for computing the addresses of BUF and HEAD from the triplet number contained in NEXT_FREE. When loading is complete, the IOP initiates transmission by writing the triplet number into the four-bit TRANSMIT control field located in IPC control word X. The IOA responds by changing the triplet state to READY.

The IOA's Sysbus Controller is continuously aware of all triplets in the READY or STAT states, as they represent data packets and status packets which can immediately be transmitted. The controller gives status packets priority over data packets, as specified by the Sysbus protocol. The Sysbus Controller maintains the priority variables, monitors Sysbus.Sender(0:7), bids according to its current priority, updates the priority variable, and (if it wins the bid) asserts its Sysbus.Sender and sequences the HEAD and BUF contents onto the Sysbus. If Sysbus.Pak remains asserted throughout the transfer, the transmitted triplet's state is set to UNACKED. If Sysbus.Nak is asserted, the Sysbus Controller captures the justification code, stops the Sysbus transfer, negates its Sysbus.Sender, writes the justification code into the rejected triplet's STAT buffer, and sets the triplet's state to ACKED.

The triplet remains in the UNACKED state until either the sysbus controller receives a status packet, or IOP software times out and aborts. The header of each incoming status packet contains the SENDER_BUF field, which identifies the triplet containing the data packet being acknowledged. The Sysbus Controller verifies that the designated triplet is in the UNACKED state, stores the incoming status packet in the triplet's STAT buffer, and then changes triplet's state to ACKED.

If the designated triplet is not in the UNACKED state, then the Sysbus Controller freezes all triplets in their present state, places an error code in the SYSBUS_ERR status field located in IPC status word X, and generates the SYSBUS_ERR interrupt. SYSBUS_ERR_INTEN resides

in IPC control register X, SYSBUS_ERR_PRI resides in IPC control register X, and SYSBUS_ERR_VECT resides in IPC control register X. The IOP can unfreeze the Sysbus Controller by asserting the UNFREEZE control bit located in IPC control register X. In general, IOP software would invoke the diagnostic subsystem to resynchonize and verify the Sysbus before UNFREEZEing.

If a parity error is detected in an incoming status packet, Sysbus.Nak is asserted, and a justification code is subsequently placed on Sysbus.Data. No triplets change state.

If one or more triplets are in the ACKED, LATENT, or RCV_HOLD states, the IOA to generates the NEW_PACK interrupt. NEW_PACK_INTEN resides in IPC control register X, NEW_PACK_LEV resides in IPC control register X, and NEW_PACK_VECT resides in IPC control register X.

IPC register X contains the NEW_AVAIL status bit, the 2-bit SITUATION status field, and the 4-bit NEW_PACKET status field. When IOP software reads this register, the Sysbus Controller first checks for any triplets in ACKED state; if one is found, NEW_AVAIL is asserted, SITUATION is set to STATUS_IN, the triplet's number is returned in NEW_PACKET, and the triplet's state is changed to XMIT_HOLD. If not, the Sysbus Controller checks for triplets in the LATENT state; if one is found, it asserts NEW_AVAIL, sets SITUATION to DATA_IN, sets the NEW_PACKET field to the triplet number, and changes the triplet's state to NOSTAT. IF no triplets are ACKED or LATENT, the Sysbus Controller checks for packets in the RCV_HOLD state; if one is found, the controller asserts NEW_AVAIL, sets SITUATION to STATUS_OUT, sets the NEW_PACKET field to the triplet number, but does not change the triplet's state. If no triplets are found in the ACKED, LATENT, or RCV_HOLD states, then NEW_AVAIL is negated.

An acknowledged triplet remains in the XMIT_HOLD state until IOP software interprets the STAT buffer and indicates it is "through" with the triplet by writing the triplet's number into the 4-bit FREE control field, located in IPC control register X. The Sysbus Controller then returns the triplet to the pool of empty buffers by changing its state to FREE. IOP software can free any triplet not presently in the READY, RECEIVING, or STAT states. IOP software can force any triplet into the LOADING state if that triplet is not presently READY, RECEIVING, or STAT. This is accomplished by writing the triplet's number into the 4-bit RESERVE control field, located in IPC register X.

IOP software can abort a triplet's transmission by writing the triplet number into the four-bit XMIT_ABORT control field located in IPC control word X. XMIT_ABORT is ignored if the triplet is not in the READY or UNACKED states. Therefore, IOP software should immediately check the triplet's state after aborting; if the state is ACKED, the triplet's STAT buffer should be examined to determine whether the abort command worked, or the data packet was transmitted and acknowledged.

4.2.2. Packet reception

The IOA's Sysbus Controller continuously monitors each packet header passing accross the Sysbus, comparing the TO_PROC field for a match. If the header's PACKET_TYPE field indicates a status packet, processing is handled as described above in packet transmission. If the incoming packet is a data packet, the Sysbus Controller must first allocate an empty buffer triplet. If none is available, it immediately asserts Sysbus.Nak and drives the appropriate justification code on Sysbus.Data during the next cycle; otherwise, it changes the empty triplet's state to RECEIVING, copies the header into the triplet's HEAD buffer, and sequentially copies the incoming packet's information field into the triplet's BUF buffer. If a Sysbus parity error is detected during reception, the controller immediately asserts Sysbus.Nak and resets the triplet's state to FREE.

When the entire data packet has been received, the Sysbus Controller changes the triplet's state to LATENT, which may generate a NEW_PACK interrupt. When the NEW_PACKET status field returns the triplet's number to the IOP, the triplet's state is changed to NOSTAT; operation of the NEW_PACK interupt and status field are described above under packet transmission.

The triplet remains in the NOSTAT state until IOP software indicates it has processed the information field contained in the triplet's BUF buffer and has fabricated a status packet in the triplet's STAT buffer. It does this by writing the triplet's number into the 4-bit STATUS control field, located in IPC control word X. The sysbus controller then changes the triplet's state to STAT, and transmits the status packet from the triplet's STAT buffer as soon as it wins Sysbus access.

If status packet transmission is completed with continuous assertion of Sysbus.Pak, the controller negates the triplet's STATUS_ERROR status bit located in IPC control word X; otherwise, the STATUS_ERROR bit is asserted. The controller then changes the triplet's state to RCV_HOLD, which may generate a NEW_PACK interrupt.

A triplet in the RCV_HOLD state is disposed of in one of three ways. If its STATUS_ERROR bit is asserted, the status packet can be retransmitted by writing the triplet's number into the STATUS control field. If a data packet response will be generated, the triplet can be reserved by writing its number into the RESERVE control field. If the transaction is complete, the triplet can be freed by writing its number into the FREE control field.

If the Sysbus Controller receives a packet header with bad parity, it asserts Sysbus.Nak and Abort.Code, and ignores all Sysbus data until the next bid cycle.

4.2.3. Sysbus Controller

The Sysbus Controller is composed of two synchronous microprogrammed state machines: the buffer microengine, and the Sysbus microengine.

The buffer microengine manages the 15 triplet state variables, interacting with both the Unibus and Sysbus microengines. The buffer microengine and packet buffers cycle twice per Sysbus cycle; during the first half, data can be transferred between the packet buffer and Sysbus microengine; during the second half, data can be transferred between the packet buffer and Unibus microengine. This time-multiplexing arrangement effectively dual-ports the packet buffers, under control of the buffer microengine.

The Sysbus microengine handles bidding, priority variable maintenance, header and information section extraction, and parity error detection.

5. Cluster Management subsystem

5.1. Control panel

The R/1000 operator control panel is located on the left front door of the mainframe. It consists of a three position power switch, and two light emitting diodes.

In power switch position A, the IOA power supply and IOPs 0 through 3 are unpowered; in positions B and C, these units are powered. Position B indicates that IOP0 is "secure" with respect to remote diagnosis; it will not answer incoming calls on the diagnostic modem, nor will it originate calls to a remote diagnostic center. Position C enables remote diagnosis. The IOA presents the REM_DIAGEN status bit in CM status word X; REM_DIAGEN is set if the power switch is in position C, and reset otherwise. No interrupt is generated on a transition of REM_DIAGEN.

The first LED indicates that the IOA power subsystem is operational. A comparator on the IOA (powered by +12 volt and -12 volt outputs "stolen" from the IOP) determines whether the IOA power supply's +5 volt output is within the range of 4.75 to 5.25 volts. If so, the LED is illuminated. Note that illumination of this LED does not imply that IOP voltages are within limits.

The second LED indicates that the IOA's diagnostic kernel controller has successfully completed its self-test. Immediately after illuminating this LED, the controller initiates verification of its console interface and the operator console. As part of this verification, the controller loops its operator console and remote diagnostic modem ports. If this loopback fails, the controller flashes the LED continuously. Thus, steady illumination of the LED

but no operator console output indicts the operator console or its cable.

5.2. Power control

Each of the R1000's four processors has its own power supply, named PS0 through PS3. CM control word X contains 3 bits for each supply: PSEx, PSMAx, and PSMBx. Setting PSEx turns on PSx's DC power; resetting PSEx turns it off. the two PSM bits select one of three output voltages: low, nominal, or high. The low and high selections are used to margin the processors during system verification. At power up, PSEx bits will be initialized to the reset state; PSMAx bits will be initialized to select nominal outputs. (Specific encodings will be specified later)

CM status word X contains one bit per processor power supply - PSV0 through PSV3 - indicating whether the output voltage of each is within the range of 4.75 to 5.25 volts. The PROC_PWR interrupt is generated on any transition of any of these four bits; this interrupt is enabled by the PROC_PWR_INTEN bit located in CM control word X. CM control word X contains PROC_PWR_VECT, the 16-bit interrupt vector for PROC_PWR. CM control word X contains PROC_PWR_PRI.

5.3. Thermal sensing

K thermal sensors are mounted at critical points within the R1000. The state of these sensors is reflected in bits Y of CM status word X. The THERMAL interrupt is generated on any transition of any of these K bits; this interrupt is enabled by the THERMAL_INTEN control bit located in CM control word X. CM control word X contains THERMAL_VECT, the 16-bit interrupt vector for THERMAL. CM control word X contains THERMAL_PRI.

5.4. Time-of-day clock

The IOA will provide a battery-powered time-of-day clock and calender which driven by a dedicated timing oscillator with accuracy of X seconds per day. The unit is expected to be used for keeping time only when the R1000 is powered down; the IOP's real-time clock, excited by power-line zero-crossings, should provide timekeeping functions during normal operations.

Specification of status word formats for determining the current time, and control word formats for specifying the current time will be deferred pending part selection. These status and control words will form a block referred to as TIME_KEEP. There may be an additional status bit indicating whether the battery voltage is within specified limits. The time-of-day clock will generate no interrupts.

5.5. Clock generation

The IOA generates the master timing signals for the R1000 cluster. Two control bits - CLKA and CLKB - located in CM control word X enable the IOP to margin the master clock with the following encodings:

CLKA	CLKB	Basic cycle (ns.)
0	0	192 (nominal)
0	1	173 (10% fast)
1	0	211 (10% slow)
1	1	clocks disabled (?)

No interrupts are generated by this logic. At power-up, both CLKA and CLKB are initialized to the reset state.

5.6. Processor Reset

CM control register X contains four control bits named RESET0 through RESET3. When asserted, these bits DC reset P0 through P3 respectively. The IOA initializes these bits to the asserted state. No interrupts are generated by this logic.

6. Diagnostic Control

6.1. Diagnostic Kernel

The Diagnostic Kernel is responsible for cluster initialization, console multiplexing, and diagnostic bus control. The Kernel on IOA0 performs these functions differently than those on IOA1 through IOA3. It is implemented with an 8031 microprocessor named the Diagnostic Kernel Controller. The kernel controller executes code from an EPROM, which effectively contains two programs: one for IOA0's kernel controller, and one for the rest. A backplane-keyed signal enables IOA0's controller to identify itself.

6.1.1. Initialization

After cluster power-up, the controller on IOA0 executes a test of itself and its interfaces. If successful, it illuminates a front-panel LED. Subsequent operator communication is via the control console (if it works). IOA0's controller then activates the bootstrap logic for IOP0, which continues the task of cluster initialization.

Kernel controllers on the other IOAs execute the same self-tests, but save the "results" for presentation (via the cluster diagnostic bus) to IOP0 during a later stage of system initialization.

6.1.2. Console multiplexing

IOA0's kernel controller multiplexes IOP0's SLU line between the control console and the diagnostic modem. The CMUX control bit, located in CM register X, specifies whether the SLU line is "connected" to the control console (CMUX negated) or the diagnostic modem (CMUX asserted). CMUX is negated during cluster powerup.

The kernel controller on IOAs 1 through 3 intercepts all SLU character traffic from their respective IOPs and routes it over the cluster diagnostic bus to IOA0s kernel controller, which forwards it to IOP0. This enables IOP0 to effectively multiplex the control console in such a way as to allow the system operator to converse with standard DEC software (e.g. diagnostics) executing on IOP1, IOP2, or IOP3.

The controller operates three full-duplex asynchronous communication line interfaces via standard MOS VLSI interfaces. These interfaces conform to EIA-RS232C, obtaining +12 volt and -12 volt power from the IOP. The control console and SLU interfaces operate at up to 9600 baud, but support no modem control. The diagnostic modem interface operates at up to 1200 baud, and supports auto originate/answer/dial modems.

6.1.3. Diagnostic bus control

The diagnostic kernel controller's integral serial port is connected to the cluster diagnostic bus.

On IOA0, the kernel controller serves as master of this multidrop bus, under the control of IOP0. The primary communication path between IOP0 and IOA0's kernel controller is the upper 512 bytes of the IOA0's BUF15, referred to as the diagnostic communication area. The kernel controller is able to "steal" packet buffer access timeslots from the Unibus microengine, and thereby directly access all 16K bytes of the packet buffer. During normal operation, the diagnostic communication area is utilized to buffer data transfers between the IOP and cluster diagnostic bus. The diagnostic kernel can generate the DIAG interrupt to IOP0 if DIAG_INTEN, a control bit located in CM register X, is asserted. DIAG_VECT, located in CM register X, supplies the vector for this interrupt, and DIAG_PRI, located in CM control register X, specifies the interrupt priority level. IOP0 can generate an interrupt to the diagnostic kernel by setting the INT_DIAG control bit, located in CM register X.

On IOAs other than 0, the diagnostic kernel is a slave with respect to the cluster diagnostic bus. Any character output by the IOP to the

SLU port is forwarded by the diagnostic kernel to IOA0. Any character received over the cluster diagnostic bus is transmitted on SLU1.

6.2. Diagnostic Interface

7. Implementation notes

7.1. IOP Interface subsystem

7.1.1. Data paths

7.1.2. Control

7.1.3. Auto-verification

7.1.4. Physical interfaces

7.2. IPC subsystem

7.2.1. Data paths

7.2.2. Control

7.2.3. Auto-verification

7.2.4. Physical interfaces

7.3. CM subsystem

7.3.1. Data paths

7.3.2. Control

7.3.3. Auto-verification

7.3.4. Physical interfaces

Table of Contents

List of Tables