```
  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 #702 for EGB   Date 3-Dec-82 1:57:21 Monitor: Rational M
File RM:<MICRO-ARCH.SYSBUS>SPEC.LPT.1, created: 21-Sep-82 9:21:24
printed: 3-Dec-82 1:57:21
Job parameters: Request created: 3-Dec-82 1:54:44   Page limit:99   Forms:NORMAL
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:A

Functional Specification of the Sysbus Interface Board

DRAFT 2

1. Summary

This document describes the complete functionality of the Sysbus Interface (SBI) board for the R1000. The purpose of this specification is to formally define the operation of the Sysbus board to a level of detail that allows microcode, hardware, and packaging designers to interface with this board correctly. The reader is presumed to be reasonably familiar with the R1000 architecture and to have access to the specifications of the other boards for explanations of their functionality.

The organization of this document is as follows; Section 2 defines Sysbus messages and introduces the microcode level protocol for interprocessor message communication along with the hardware resources necessary for this communication. Section 3 provides a detailed definition of the functionality, on a block by block basis, of each block on the attached block diagram. Section 4 defines the Sysbus board microword along with its encodings. Section 5, discusses the details of the microcode level transfer protocol and, along with the previous section, defines the microcode interface to the Sysbus board by specifying what hardware resources are available to the microcoder and the restrictions that are placed on these resources. Section 6 discusses the diagnostic strategies that are employed to debug the board at both the hardware and microcode levels and what hardware support is available to support these strategies. Finally, section 7 details the issues that concern the hardware and packaging designers when interfacing to the Sysbus board. These issues include timing considerations, chip count and power estimates, and board layout details.

2. Sysbus Messages and Message Transfers

Sysbus transfers occur as the result of 3 distinct types of processor activity: cross-processor memory access, cross-processor package or task elaboration and cross-processor entry calls. Each of these types of transfers is mapped onto explicit Sysbus messages by the microcode and hardware. The maximum number of devices that can be addressed on a Sysbus is eight. This corresponds to a fully configured R1000 cluster with four Sysbus addresses occupied by R1000 processors and four addresses occupied by I/O adapters (IOA's). Table 2-1 lists each device and its address on the sysbus. There is no distinction made in the Sysbus protocol between processors and IOA's. In the remainder of this document, when a reference is made to a processor, it also applies to an IOA unless otherwise noted.

The term "home processor" is used in the remainder of this document whenever a sysbus device (processor or IOA) is refering to itself. A devices "home processor number" corresponds to its address on the sysbus (e.g. processor 0's home processor number is 4).

Table 2-1: Sysbus Addresses

Sysbus Address	Sysbus device
0	I/O adapter 0 (IOA0)
1	I/O adapter 1 (IOA1)
2	I/O adapter 2 (IOA2)
3	I/O adapter 3 (IOA3)
4	Processor 0 (P0)
5	Processor 1 (P1)
6	Processor 2 (P2)
7	Processor 3 (P3)

2.1. Messages

There are two kinds of sysbus messages. The first kind, a _sysbus data message_, is a variable sized data transfer between two processors in an R1000 cluster. In order to meet hardware timing restrictions, these messages are subdivided into packets. A packet is made up of two parts; the header and the data.

The header is a maximum of 8 words (one word equals 128 bits) long and contains the following information: a from-processor ID (FROM_PROC), a to-processor (TO_PROC), the size of the header in half-words(64 bits) (HEADER_LENGTH),the size of the packet in half-words (PACKET_LENGTH), the type of the packet (PACKET_TYPE), possibly a from-task-id, a to-task-id, and various amounts of other information depending on what type the packet is. The data part of a packet is a maximum of 64 words long.

A data message is made up of one or more packet transfers. Each packet of a data message may be a short packet; there is no hardware restriction on the length of any packet.

The second kind of Sysbus message is a _sysbus status message_. This is a one half-word message that is used to verify sysbus data messages. Further discussion of status messages is given in section 2.3 of this document.

2.2. Buffer Resources

Each processor contains a receive buffer and a transmit buffer for each of the other addresses in the system making a total of 8 transmit and 8 receive buffers on each processor. The receive buffers are used to receive packets from each of the other processors of the sytem; the transmit buffers are used to send packets to each of the other processors. All transmit and receive buffers are exactly 64 words long.

2.2.1. Receive Buffers

The receive and transmit buffers are numbered to correspond with device addresses on the Sysbus, from 0 to 7. The receive buffer corresponding to the home processor (e.g. receive buffer 4 for processor 0) is divided into 8 sections called receive header sections. The receive header sections are numbered from 0 to 7 to correspond to the eight Sysbus addresses. Each section is 8 words long. When a processor receives a packet from device N, the hardware puts the header part of the incoming packet into the receive header section corresponding to that device (i.e. section N) and the data part of the packet into the corresponding receive buffer.

The receive header section corresponding to the home processor (e.g. receive header section 4 for processor 0) is also divided into 8 sections called receive status sections. Each of these sections contains a one half-word status code that is used to send HOME_PROCESSOR status to each of the other processors, and one half-word that is not used. The contents of each of these receive status sections is loaded by the microcode with the status of the packet last received from the corresponding processor. Further discussion of how packet status is communicated between processors is given in Section 5 of this document.

2.2.2. Transmit Buffers

The eight transmit buffers are organized very much like the receive buffers. The transmit buffer corresponding to the home processor is divided into eight sections called transmit header sections. The transmit header sections are numbered from 0 to 7 to correspond to the eight sysbus addresses. Each section is 8 words long. When a processor sends a packet to device N, the processor microcode puts the header for the outgoing packet into the transmit header section corresponding to that device (i.e. section N) and the packet data into the corresponding transmit buffer.

The transmit header section corresponding to the home processor (e.g. transmit header section 4 for processor 0) is also divided into 8 sections called transmit status sections. Each of these sections contains a one half-word status code from each of the other processors in the system, and one half-word that is not used. These codes indicate the status of the last packet that was sent to the corresponding processor. Further discussion of how packet status is communicated between processors is given in Section 5 of this document.

2.3. Sysbus Transfers

A normal sysbus transfer consists of the following sequence:

1. Source processor loads header and packet into transmit buffer for destination processor.

2. Header and data are sent to the corresponding receive buffer on the destination processor.

3. A status response is sent back to the source buffer indicating the disposition of the packet.

Sysbus transfers are handled by hardware from the time the packet is fully loaded into the transmit buffer until it is completely moved into the appropriate receive buffer. All other packet handling is performed by microcode.

Each processor maintains three status bits for each of the other 7 processors in the system. These bits are:

SYSBUS_RECEIVE Indicates that the receive buffer for the corresponding processor is not empty.

SYSBUS_TRANSMIT Indicates that the transmit buffer for the corresponding processor is not empty, or that a status reponse has not yet been received for the last packet sent.

STATUS_RECEIVE Indicates that a status message was received from the corresponding processor.

When processor S (source processor) sends a packet to processor D (destination processor), the microcode on S loads the packet into the transmit buffer and transmit header section that correspond to D, then sets the SYSBUS_TRANSMIT bit that corresponds to D. The hardware sends the packet to the receive buffer on processor D that corresponds to S, sets S's SYSBUS_RECEIVE bit on D, and generates a NEW_PACKET micro event on D. A TRANSFER_COMPLETE micro-event is generated on processor S. This event will probably be disabled, but is implemented to allow double-buffering of transmit buffers. Since more transfers could have been initiated before the TRANSFER_COMPLETE is generated, the processor number of D can be read in the micro-event handler.

The NEW_PACKET micro-event does not currently perform any major function. It exists primarily to provide flexibility for future implementations and improvements of the transfer protocol. Examination of the SYSBUS_RECEIVE bits and processing of the receive buffers is deferred until the SYSBUS_PACKET macro event. Posting of the SYSBUS_PACKET event is performed by the NEW_PACKET micro-event.

When macro-events are next enabled (either at a dispatch or explicitly by microcode in some very long microroutine), the SYSBUS_PACKET event handler examines the SYSBUS_RECEIVE bits to determine which processor(s) it has received packets from. When the packet in receive buffer S is sufficiently processed to determine the appropriate response to the message, the SYSBUS_RECEIVE bit corresponding to that buffer is cleared and a one half-word status code is written into the RECEIVE_STATUS section that corresponds to processor S. The microcode on D then issues a SEND_STATUS command and the hardware moves the status code from the receive header section on D into the appropriate TRANSMIT_STATUS section on S.

When the hardware on processor S receives the status message from D, it sets the STATUS_RECEIVE bit corresponding to D and generates a NEW_STATUS micro-event. The NEW_STATUS event handler does not currently perform any major function. It exists primarily to provide flexibility for future implementations and improvements of the transfer protocol. Examination of the STATUS_RECEIVE bits and processing of the status message code is deferred until the SYSBUS_STATUS macro event. Posting of the SYSBUS_STATUS event is performed by the NEW_STATUS micro-event handler.

When this macro-event is taken by S, the SYSBUS_STATUS handler will determine the number of the processor which responded and examine the status code that was sent. When the status code for D is processed, the microcode on S resets the SYSBUS_TRANSMIT bit corresponding to processor D to indicate that D's transmit buffer is no longer busy, this Sysbus transaction is complete and the buffer may be used to transmit another packet to D.

2.4. Error Handling

The Sysbus control logic and microcode can detect and respond to various errors which occur during a Sysbus transfer. The primary mechanism for this is a hardware Negative Acknowledge (NAK) signal between nodes on the Sysbus. Both transmit and receive nodes will constantly monitor NAK during a transfer. If the either node detects an error (such as parity or incorrect message length) it will assert NAK and abort transfer activity. A set of error identification lines will be driven with a code indicating the nature of the error. The sending processor will then generate a TRANSFER_COMPLETE micro-event with an error flag set, and await a status response from the destination node, if NAK was activated by the receiver. This response will describe the error in greater detail, which will allow the microcode to make a decision whether to retry the transmission.

Timeouts are at the discretion of the microcode. These can be implemented either by proper setup of one of the on-board timers, or through the use of microcode loops. No timing checks are made by the hardware during bus arbitration or message transfer. However, if the receiving processor does not respond with a positive acknowledgement

(PAK) within one bus cycle of transmission of the last word of the packet, the TRANSFER_COMPLETE (with error) micro-event will be issued. Since an error of this type could possibility preclude the receiving processor from issuing a status response, the microcode should timeout if the TRANSFER_COMPLETE micro-event is disabled.

During bus transmissions the hardware is also monitoring a set of check-lines on the bus to determine if any other node is attempting to drive the bus. If it is detected that there is bus contention, the transfer will be immediately aborted, and a TRANSFER_COMPLETE (with error) micro-event generated.

3. Block Diagram Functional Definition

This section references the block diagram of the Sysbus board attached to this document. The functionality of each block in the diagram is discussed in detail in the following sections.

3.1. Sysbus

The Sysbus is the medium through which all interprocessor communication and data pass. Input/Output operations also are performed over the Sysbus, since the I/O adapters are considered as processors for the purpose of this description. The Sysbus is etched on the R1000 backplane, and is connected only to the Sysbus Interface board of each processor and each I/O adapter. It is driven by standard Schottky tri-state TTL(data lines), and TTL open-collector (control lines) Termination is supplied by plug-ons to the backplane (if needed). Etch lengths are kept to an absolute minimum on each board, so that a calculated backplane trace impedance of 92 ohms should yield little impedance mismatch (and associated reflections and ringing). Logically the Sysbus consists of:

64 signal lines
8 byte parity lines
(8 priority arbitration/processor identification lines)*
1 status priority line
1 negative acknowledgment (NAK) line
1 positive acknowledgment (PAK) line
1 BUS_BUSY line
2 error-id lines

* Priority lines may have to be timeshared on the signal lines, in which case a 3 line processor id scheme would be used.

Processor addresses are embedded in the packet header data and decoded by hardware.

A processor can gain control of the Sysbus by exercising a successful bid. Arbitration is performed via a modified rotating daisy chain method. This method increments the priority of each processor at each bid. A successful bidder has his priority rotated to zero (lowest), and the non-bidding processor with the highest priority is changed to a priority that is one greater the winning bidder. This allows non-bidding processors with high priorities to remain relatively high, while also giving lower priority processors an opportunity to move up. Bidding can occur in any cycle when the bus is not busy. The winning processor is obligated to drive the BUS_BUSY line the cycle immediately following the bid, or else bus contention could result.

One extra bid line is provided for high priority responses. It should only be used by processors attempting to access the bus for a status response to a previously transmitted message. If activated by one or more processors, all bidders not sending status responses are removed from the bid lines, and arbitration is only among the processors driving the status bid line. This mechanism allows fast completion of macro-level transactions on a heavily loaded system.

Once a processor has control of the bus, it immediately begins sending header information. It also activates the BUS_BUSY line, and drives its' decoded physical processor number onto the priority lines. Although a processor can drive any of the priority lines during a bid cycle depending on his rotated priority, the processor number is constant during the powered-up life of the R1000. Thus, by constantly monitoring these lines during a bus transaction, a processor will be able to detect one of several error conditions. If a processor in control of the bus sees any but its' own line activated, it will immediately terminate the transfer and report an error to a micro-handler.

Processors must also monitor the Negative Acknowledgment (NAK) line during the transfer. If either node detects a parity error, incorrect length, or some other error, it will activate NAK. This will abort bus activity, and a 2 bit code will be placed on the error id lines. If possible, the detecting node will prepare a status message with more information on the error. If the transfer terminates normally, Positive Acknowledgment (PAK) should be asserted. The source processor will expect PAK to be active one cycle following the last word transfer, and will report an error to the microcode if otherwise.

The 8 priority lines, the status priority line, NAK, PAK, and BUS_BUSY are all open-collector lines driven by all processors in the system.

3.2. Sysbus Buffer

The principal resource for storing and receiving data on the Sysbus is the Sysbus Buffer. The buffer is 1024 words long and 128 bits wide, constructed of 32 1KX4 static memory chips. Its' bidirectional I/O lines can be accessed by either the processor's TYPE and VAL buses, or the Sysbus itself. Reads and writes from the Sysbus, and reads to the TYP and VAL buses are pipelined through one level of registers. Writes to the buffer from the TYPE and VAL buses are performed directly.

The buffer can be accessed by both the Sysbus and the processor in the same cycle. This is accomplished by dividing each cycle into two parts, with the Sysbus controlling the buffer in the first half, and the processor in control during the second. Any combination of reads and writes can be done, with Sysbus data transferred via registers, and processor read data loaded into a register. Standard R1000 error checking is performed on all data to and from the processor. On transfers to the Sysbus, parity is checked or generated on each byte.

Since the buffer is 128 bits wide, and the Sysbus only contains 64 bits, each buffer read or write requires two Sysbus cycles. This alternating access is performed automatically by the hardware.

3.2.1. Buffer Addressing

The buffer is initially divided into 2 halves of 512 words each: one for received data and one for transmitted data. Each of these halves is further subdivided into individual buffer spaces of 64 words for each of the 8 processors in the system. The buffer assigned to each processor itself (the home processor) is used for storage of header information. Within this header area, 8 words are allocated to each processor in the system. The home processor's area is used for status words for each other processor. For each type of access, the buffer address breaks down as follows:

$T/\overline{R}$	Remote Proc # (3 Bits)	Buffer Word # (6 Bits)
0	1	4 ... 9

This format addresses a specific word in the transmit or receive buffer of a given processor.

$T/\overline{R}$	Home Proc # (3 Bits)	Remote Proc # (3 Bits)	Header Word # (3 Bits)
0	1	4	7 ... 9

This format is used for addressing a header word for a remote processor. The Home processor number is set in hardware and is not normally accessible to the microcode.

```
 -------------------------------------------------
| T/ |  Home Proc # |  Home Proc # | Remote Proc #|
| R‾ |   (3 Bits)   |   (3 Bits)   |  (3 Bits)    |
 -------------------------------------------------
0    1              4              7            9
```

This third format addresses the status of a remote processor.

These addressing formats are transparent to the microcode, which simply specifies which type of buffer storage to access (data, header, or status), a processor number (which is set in a register), and whether to address transmit or receive information. For the first two formats, an address offset counter must also be set to an initial value, from which it can be incremented to access successive locations.

The buffer can also be addressed directly by an address counter of 10 bits, which is settable under microcode control from the VAL bus. This counter can then be incremented as the buffer is accessed. This feature would normally be used for diagnostic purposes only.

3.3. Sysbus Control

3.4. Bus Interfaces

3.4.1. Val and Type Busses

The Sysbus connects to the Val and Type buses via transceivers which also link to Internal Val and Internal Type buses. These buses are used as input to the error correction circuit, as I/O to the dummy RDR, and as I/O for buffer data. During reads from the buffer to the Internal buses, a pipeline register is used. Data to be written into the buffer memory is driven directly from the IVAL and ITYPE buses onto the buffer's bidirectional I/O lines.

3.4.2. Sysbus

Since the Sysbus is 64 bits wide and the buffer contains 128 bits, data must be registered in and out of storage. Two 64 bit registers capture data read out of the buffer and drive out onto a 64 bit internal bus (SOBUS). The SOBUS is driven directly out onto the SYSBUS. Data coming in from the Sysbus is placed onto another internal bus (SIBUS) which is driven onto the low or high 64 bits of the buffer's I/O lines. Parity is checked on data from the Sysbus when it is the SIBUS. A separate 8 bit parity register sits on the Sysbus for latching the parity bits for checking.

3.5. Timers

3.6. Error Checking and Correction

3.6.1. Error Checking Theory

The error code implemented on the Sysbus board is a modified Hamming code that can correct single-bit errors and detect all double-bit errors (and some other multi-bit errors) on a 128-bit word. This requires an extra 9 bits to be added to the word in storage for check code bits. When a word is written to memory, the 128-bit value presented on the TYPE and VAL buses is parity-checked over 9 different groups of 64 bits each. These 9 parity values become the check-bits to be stored into memory. When read out of memory, the same bits are parity-checked again, and the result 9 check-bits compared against those read from memory. If the two values match, there is no error, but if different an error exists. The parity of the exclusive-OR of the two values (called the syndrome) indicates whether the result is a single-bit or double-bit error. The syndrome of a single-bit error can be decoded to indicate which bit is in error. This result is then passed to the microcode, which corrects the bad bit and rewrites the good value to memory, as well as passing it on to the requesting microroutine. A double-bit error is flagged as uncorrectable, and a machine check results.

Tables 3-1 through 3-4 detail the bits used in the generation of check bits.

3.6.2. Error Correction Implementation

Error correction check bits are generated by running the data bits through two levels of parity generators. In theory, each data bit will go to either 3 or 5 different parity groups. However, due to the fact that the code is designed to yield byte parity, and careful assignment of bits, several of the intermediate parity terms can be used more than once. Thus, the first level of parity generators for each check bit may vary from 4 to 8 chips each.

Sixteen of the intermediate parity terms provide byte parity for the VAL and TYPE buses, which are driven out to the backplane on bus sourcing cycles, or compared to the driven bits during bus reads. The ECC check bits are driven to the check bit bus on memory writes. On a memory read, the check bits are actually syndrome bits (Old_check_bits XOR New_check_bits), and are input to a PROM and a NAND checker. If any syndrome bit is non-zero, an ECC error is indicated by the NAND

Table 3-1: Error Correction - VAL Bits 0:31

VAL Data Bit # | Check Bit #	0	1	2	3	4	5	6	7	8
0			X					X	X
1			X	X					X
2			X			X			X
3			X		X				X
4			X				X	X	
5			X			X		X	
6			X			X	X		
7			X		X			X	
8				X				X	X
9				X			X		X
10				X			X	X	
11				X		X			X
12				X	X			X	
13				X	X				X
14				X	X	X			
15			X	X				X	
16						X		X	X
17						X	X		X
18					X	X	X		
19					X	X			X
20					X	X		X	
21			X		X	X			
22						X	X	X	
23			X	X		X			
24				X	X		X		
25					X		X	X	
26			X		X		X		
27					X		X		X
28			X				X		X
29				X		X	X		
30			X	X			X		
31							X	X	X

(X indicates data bit participates in XOR generation of check bit)

Table 3-2: Error Correction - VAL Bits 32:63

VAL Data Bit # | Check Bit #	0	1	2	3	4	5	6	7	8
32	X	X					X	X	X
33	X	X				X		X	X
34	X	X				X	X		X
35	X	X	X				X		X
36	X	X	X	X					X
37	X	X		X				X	X
38	X	X		X			X		X
39	X	X		X		X			X
40	X	X	X	X	X				
41	X	X			X		X		X
42	X	X	X		X			X	
43	X	X		X	X				X
44	X	X	X		X				X
45	X	X		X	X			X	
46	X	X		X	X		X		
47	X	X	X		X		X		
48	X	X	X			X			X
49	X	X	X			X		X	
50	X	X	X			X	X		
51	X	X		X	X	X			
52	X	X			X	X	X		
53	X	X			X	X		X	
54	X	X	X		X	X			
55	X	X	X	X		X			
56	X	X	X					X	X
57	X	X	X				X	X	
58	X	X		X			X	X	
59	X	X	X	X				X	
60	X	X				X	X	X	
61	X	X			X		X	X	
62	X	X			X			X	X
63	X	X		X		X		X	

Table 3-3: Error Correction - TYPE Bits 0:31

TYPE Data Bit # / Check Bit #	0	1	2	3	4	5	6	7	8
00		X	X	X	X		X		
01		X	X	X	X			X	
02		X	X	X			X	X	
03		X	X	X			X		X
04		X	X	X				X	X
05		X	X	X		X		X	
06		X	X	X		X	X		
07		X	X	X	X	X			
08		X		X	X	X	X		
09		X		X	X		X		X
10		X		X	X			X	X
11		X		X	X	X			X
12		X		X	X		X	X	
13		X	X		X			X	X
14		X	X	X	X				X
15		X		X	X	X		X	
16		X		X		X	X	X	
17		X		X		X	X		X
18		X	X		X	X		X	
19		X	X		X	X			X
20		X	X			X		X	X
21		X		X		X		X	X
22		X	X		X	X	X		
23		X			X	X		X	X
24		X			X	X	X	X	
25		X	X			X	X		X
26		X	X			X	X	X	
27		X	X				X	X	X
28		X			X		X	X	X
29		X		X			X	X	X
30		X	X		X		X		X
31		X			X	X	X		X

Table 3-4: Error Correction - TYPE Bits 32:63

TYPE Data Bit #	Check Bit # 0	1	2	3	4	5	6	7	8
32	X		X	X	X		X		
33	X		X	X	X			X	
34	X		X	X			X	X	
35	X		X	X			X		X
36	X		X	X				X	X
37	X		X	X		X		X	
38	X		X	X		X	X		
39	X		X	X	X	X			
40	X			X	X	X	X		
41	X			X	X		X		X
42	X			X	X			X	X
43	X			X	X	X			X
44	X			X	X		X	X	
45	X		X		X			X	X
46	X		X	X	X				X
47	X			X	X	X		X	
48	X			X		X	X	X	
49	X			X		X	X		X
50	X		X		X	X		X	
51	X		X		X	X			X
52	X		X			X		X	X
53	X			X		X		X	X
54	X		X		X	X	X		
55	X				X	X		X	X
56	X				X	X	X	X	
57	X		X			X	X		X
58	X		X			X	X	X	
59	X		X				X	X	X
60	X					X	X	X	X
61	X			X			X	X	X
62	X		X		X		X		X
63	X				X	X	X		X

check, and the PROM outputs indicate the bit_in_error for a single-bit error. One of the PROM outputs shows that a multiple bit error has occurred, unless both the bit_in_error is 127 (all 1's) and the multi-bit error signal is true, in which case a check-bit error has occurred. Check-bit errors are not corrected, but instead the data word is rewritten with a new check code.

Any of the error types will generate a micro-event. This event can be disabled by the microcode, such as when the dummy RDR or the CSA is being sourced to the buses.

A diagnostic feature is available to allow the microcode to specify the 9 bit check code to be written to memory. This can be used for memory tests or generating known memory errors. The register for this feature is loadable from the VAL bus.

3.7. Clock Distribution

4. Microword Description

4.1. Microword Field Definition

The microword used to control the SYSBUS appears as follows:

ADDR_MODE (2 bits)

00	Index by LOCAL_PROC register
01	Direct
10	Priority
11	Undefined

DATA_GROUP (3 bits)

000	Buffer
001	Header
010	Status Word
011	Flag display
100	Flag set
101	Flag clear
110	STATUS_RESPONSE flag
111	NOP

ACCESS_MODE (1 bit)

0	Read or Receive (see context notes)
1	Write or Transmit (see context notes)

GENERAL_CONTROL (5 bits)

00000		Read HOME_PROC value
00001		Write HOME_PROC value (diagnostic function only)
00010		NOP
00011		NOP
00100		Read LOCAL_PROC value
00101		Write LOCAL_PROC value
00110		Increment LOCAL_PROC value
00111		NOP
01000		Read BUFFER_ADDR_REG
01001		Write BUFFER_ADDR_REG
01010		Increment BUFFER_ADDR_REG
01011		Decrement BUFFER_ADDR_REG
01100		Clear Sysbus Packet Event
01101		Clear Sysbus Status Event
01110		Clear Slice Timer Event
01111		Clear GP Timer Event
10000		Load Slice Timer
10001		Read Slice Timer
10010		Enable Slice Timer
10011		Inhibit Slice Timer
10100		Load GP Timer
10101		Read GP Timer
10110		Enable GP Timer
10111		Inhibit GP Timer
11000		Read Micro-event Mask
11001		Load Micro-event Mask
11010		NOP
11011		NOP
11100		Load Check-bit Register
11101		NOP
11110		Send STATUS_RESPONSE
11111	def	NOP

TYPE_VAL_BUS_SOURCE (4 bits)

	TYPE_BUS_SOURCE	VALUE_BUS_SOURCE
0000	TYPE board	VALUE board
0001	TYPE board	FIU board
0010	FIU board	VALUE board
0011	FIU board	FIU board
0100	MEMORY board	MEMORY board
0101	SYSBUS board	SYSBUS board
0110	MICROSEQUENCER board	MICROSEQUENCER board
0111	TYPE board	MEMORY board
1000	FIU board	MEMORY board
1001	NOP	
1010	NOP	
1011	NOP	

1100	NOP
1101	NOP
1110	NOP
1111	all boards disabled from driving the TYPE, VA FIU busses

FIU_BUS_SOURCE (2 bits)

00	FIU board
01	VALUE board
10	TYPE board
11	MICROSEQUENCER board

ADDR_BUS_SOURCE (2 bits)

00	FIU board
01	VALUE board
10	TYPE board
11	MICROSEQUENCER Board

LOAD_WDR (1 bit)

BREAK_POINT (1 bit)

4.2. Microword Field Context

Following are the interpretations of the ACCESS_MODE and ADDR_MODE fields for each of the group contexts:

Buffer Space (000)

ACCESS_MODE=0: Read from buffer
ACCESS_MODE=1: Write to buffer
ADDR_MODE=Index (00): Address buffer using LOCAL_PROC register. Writes occur to the associated transmit buffer, reads from the receive buffer.
ADDR_MODE=Direct (01): Address buffer using BUFFER_ADDR_REGISTER. Writes or reads may be to any location.
ADDR_MODE=Priority (10): N/A. Undefined.

Header Block (001)

ACCESS_MODE=0: Read from receive header
ACCESS_MODE=1: Write to transmit header
ADDR_MODE=Index (00): Address header of LOCAL_PROC
ADDR_MODE=Direct (01): N/A
ADDR_MODE=Priority (10): N/A

Status Word (010)

ACCESS_MODE=0: Read received status word

ACCESS_MODE=1: Write transmitted status word
ADDRESS MODE=Index (000): Access status of LOCAL_PROC

ADDR_MODE=Direct (001): N/A
ADDR_MODE=Priority (010): N/A

Flag display (011)

ACCESS_MODE=0: Access receive flag
ACCESS_MODE=1: Access transmit flag
ADDR_MODE=Index (000): Place status bit of LOCAL_PROC onto VAL(63)
ADDR_MODE=Direct (001): Place status bits of all processor buffers onto VAL(56:63)
ADDR_MODE=Priority (010): Place status bit of highest numbered active processor onto VAL(60), and processor number onto VAL(61:63). If no processor has an active flag, VAL(60)=0 and VAL(61:63) are indeterminate. ACCESS_MODE=1 will display number of highest processor generating TRANSFER_COMPLETE. VAL(57) indicates an error condition, with VAL(58:59) containing the error code.

Flag set (100)

ACCESS_MODE=0: Access receive flag
ACCESS_MODE=1: Access transmit flag
ADDR_MODE=Index (000): Set status bit of LOCAL_PROC
ADDR_MODE=Direct (001): Set status bits of all processors to value on VAL(56:63)
ADDR_MODE=Priority (010): N/A

Flag clear (101)

ACCESS_MODE=0: Access receive flag
ACCESS_MODE=1: Access transmit flag
ADDR_MODE=Index (000): Set status bit of LOCAL_PROC
ADDR_MODE=Direct (001): Set status bits of all processors to value on VAL(56:63)
ADDR_MODE=Priority (010): N/A

STATUS_RESPONSE flag (110)

ACCESS_MODE=0: Read flag(s)
ACCESS_MODE=1: Clear flag(s)
ADDR_MODE=Index (00): Access STATUS_RESPONSE flag of LOCAL_PROC
ADDR_MODE=Direct (01): Access STATUS_RESPONSE flags of all processors. Reads place the flags onto the Low-order bits of VAL.
ADDR_MODE=Priority (10): No action if ACCESS_MODE=1. If ACCESS_MODE=0, places STATUS_RESPONSE flag of highest numbered active processor onto VAL(60) and processor number on VAL(61:63). If there is no active processor, VAL(60) will be 0.

5. Microcode Usage

5.1. Buffer Operations

The primary method of accessing the buffer will be with the Index Mode. For transmits, the LOCAL_PROC register would be loaded from the VAL bus with a processor number determined by a microroutine which translated a Virtual Processor ID to a physical processor. Header information would then be loaded with GROUP=Header and Index Mode. The individual words in the header would be indicated by the value in the Buffer_Address_Register, which can be loaded and incremented by a General_Control micro-order. The packet data can then be loaded in the same manner with Group=Buffer_Space. When the packet is fully entered, it can then be transferred by ACCESS=Transmit, GROUP=Flag Set, which will cause the Sysbus logic to begin the interprocessor transfer. When it is determined that the transfer is complete, either through the TRANSFER_COMPLETE micro-event or a SYSBUS_STATUS macro-event, GROUP=Flag_Clear will reset the transmit-busy flag.[Note: After a SYSBUS_STATUS event, the STATUS_RESPONSE flag should be checked to verify which processor responded, especially if more than one transmit had been initiated.]

When a SYSBUS_PACKET macro event is taken, the GROUP=Flag_Display with Priority Mode can be used to determine which processor has sent a packet. This processor number can then be loaded into the LOCAL_PROC register and the header and buffer read out. When the data has been processed, ACCESS=Transmit, GROUP=Status will accept the return status. This status will then be sent back to the source processor by a GENERAL_CONTROL micro-order.

The TRANSFER_COMPLETE micro-event handler must determine which processor generated the event by using the GROUP=Display flag in Priority Mode with ACCESS=1. The display will also indicate whether the transfer terminated normally or not, and give a 2 bit error code if there was an abort.

Table of Contents

1. Summary 1
2. Sysbus Messages and Message Transfers 1
    2.1. Messages 2
    2.2. Buffer Resources 2
        2.2.1. Receive Buffers 3
        2.2.2. Transmit Buffers 3
    2.3. Sysbus Transfers 4
    2.4. Error Handling 5
3. Block Diagram Functional Definition 6
    3.1. Sysbus 6
    3.2. Sysbus Buffer 7
        3.2.1. Buffer Addressing 8
    3.3. Sysbus Control 9
    3.4. Bus Interfaces 9
        3.4.1. Val and Type Busses 9
        3.4.2. Sysbus 9
    3.5. Timers 10
    3.6. Error Checking and Correction 10
        3.6.1. Error Checking Theory 10
        3.6.2. Error Correction Implementation 10
    3.7. Clock Distribution 15
4. Microword Description 15
    4.1. Microword Field Definition 15
    4.2. Microword Field Context 17
5. Microcode Usage 19
    5.1. Buffer Operations 19

List of Tables

Table 2-1: Sysbus Addresses 2
Table 3-1: Error Correction - VAL Bits 0:31 11
Table 3-2: Error Correction - VAL Bits 32:63 12
Table 3-3: Error Correction - TYPE Bits 0:31 13
Table 3-4: Error Correction - TYPE Bits 32:63 14